



2ND EDITION

# AWS Security Cookbook

Practical solutions for securing AWS cloud infrastructure  
with essential services and best practices

A decorative orange geometric shape, resembling a stylized 'L' or a corner bracket, located in the bottom left corner of the cover.

HEARTIN KANIKATHOTTU

# **AWS Security Cookbook**

Practical solutions for securing AWS cloud infrastructure with essential services and best practices

**Heartin Kanikathottu**



# AWS Security Cookbook

Copyright © 2024 Packt Publishing

*All rights reserved.* No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

**Group Product Manager:** Dhruv Jagdish Kataria

**Publishing Product Manager:** Prachi Sawant

**Book Project Manager:** Ashwin Kharwa

**Senior Editor:** Mohd Hammad

**Technical Editor:** Irfa Ansari

**Copy Editor:** Safis Editing

**Proofreader:** Mohd Hammad

**Indexer:** Subalakshmi Govindhan

**Production Designer:** Jyoti Kadam

**DevRel Marketing Coordinator:** Marylou De Mello

First published: February 2020

Second edition: October 2024

Production reference: 1220824

Published by Packt Publishing Ltd.

Grosvenor House

11 St Paul's Square

Birmingham

B3 1RB, UK

ISBN 978-1-83508-189-1

[www.packtpub.com](http://www.packtpub.com)

*To my wife, Sneha, and daughters, June and Novanah, for their sacrifices and support.  
To the Cloudericks and Coding Architect CA communities for their support. Most importantly,  
to God for the opportunity to work on this book.*

*– Heartin Kanikathottu*



# Contributors

## About the author

**Heartin Kanikathottu** is an accomplished cloud architect renowned for leading technological transformations in cloud computing and security at prestigious organizations. He is also a prolific author recognized globally, with his book, *AWS Security Cookbook, First Edition*, being named the eighth best in cloud computing by *BookAuthority.org* in 2020. His impressive career includes roles as founder of Trainso and Coding Architect Canada, vice president at Morgan Stanley, principal architect at Societe Generale, and cloud and security architect at VMware. He has also worked at TCS, SAP Ariba, and IG Group. He holds over 15 professional certifications from Microsoft, Amazon, Oracle, Pivotal, and IBM, and dual master's degrees in cloud computing and data analytics. He is also a regular speaker at many technical forums.

*I want to thank God for everything. I am deeply grateful to my wife, Sneha, my daughters, June and Novanah, and my parents, Gresamma and Jacob, for their love and support. Special thanks to Dilip, Harsha, Sarah, and Angeline for their support and care. I also want to thank the Cloudericks and Coding Architect CA study groups and students, especially Elsa, Itegebe Michael, Srihari, Aswini, Krishnaveni, and Abhijeet, for their support with the book.*

## About the reviewers

**Elsa Sherine Stephen** is an accomplished IT professional with over 17 years of experience in project management and Agile methodologies. She holds a B.Tech and M.Sc in computer science and began her career at Infosys, where she progressed through roles from developer to senior project manager. Elsa has a proven track record of setting up and leading high-performing teams, delivering complex projects on time and within budget. Her expertise includes managing mixed-tech stack portfolios and interacting directly with clients and stakeholders to ensure seamless project execution.

*I would like to extend my heartfelt gratitude to Packt Publishing for the opportunity to be a part of this project. A special thanks to Heartin, the author and a good friend of mine, for his insightful and comprehensive work on AWS security. I am also deeply grateful to the cloud community for their continuous support and knowledge sharing, which has been invaluable in this journey.*

**Sabari Sundaram** is a seasoned IT security professional with over 14 years of experience in cloud security, threat detection, and incident response. Throughout her career, she has developed and implemented robust security programs, including security incident response and automation, for both on-premises and cloud-based environments. During her role at AWS, she has played a vital part in driving the launch of some AWS security services and features.

**Sneha Thomas** is a senior software engineer with over 13 years of experience in the IT industry. She currently serves as a project delivery lead at MPhasis, Canada. Sneha has contributed her expertise to organizations such as the City of Calgary, ANZ, Publicis Sapient, Ericsson, TCS, and Cognizant. As a lead software engineer, she has extensive experience with technologies such as Java, Spring, Hibernate, Angular, and OpenShift, and is proficient in platforms such as Jenkins, Udeploy (now known as IBM UrbanCode Deploy), and Codefresh, with substantial AWS knowledge. Sneha holds a master's in cloud computing and a bachelor's in electronics and communications. She was also the technical reviewer of *Serverless Security Cookbook* and the first edition of *AWS Security Cookbook*, published by Packt Publishing.



# Table of Contents

## Preface

xv

## 1

### Setting Up AWS Accounts and Organization 1

Technical requirements	2	How to do it...	15
Setting up IAM, account aliases, and billing alerts	3	How it works...	21
Getting ready	3	There's more...	23
How to do it...	3	See also	25
How it works...	11	User management and SSO with IAM Identity Center	25
There's more...	12	Getting ready	25
See also	14	How to do it...	26
Multi-account management with AWS Organizations	14	How it works...	37
Getting ready	14	There's more...	40
		See also	41

## 2

### Access Management with IAM Policies and Roles 43

Technical requirements	44	Using policy variables within IAM policies	56
Creating a customer-managed IAM policy	45	Getting ready	57
Getting ready	45	How to do it...	57
How to do it...	45	How it works...	59
How it works...	51	There's more	59
There's more...	54	See also	60
See also	56		

<b>Creating customer-managed policies in IAM Identity Center</b>	<b>60</b>	How to do it...	71
Getting ready	61	How it works...	74
How to do it...	62	There's more...	74
How it works...	67	See also	74
There's more...	68	<b>IAM cross-account role switching and identity account architecture</b>	<b>75</b>
See also	68	Getting ready	76
<b>Setting IAM permission boundaries for IAM entities</b>	<b>68</b>	How to do it...	76
Getting ready	68	How it works...	85
How to do it...	69	There's more...	86
How it works...	70	See also	87
There's more...	71	<b>Cross-service access via IAM roles on EC2 instances</b>	<b>87</b>
See also	71	Getting ready	87
<b>Centralizing governance in AWS Organizations with SCPs</b>	<b>71</b>	How to do it...	88
Getting ready	71	How it works...	90
		There's more...	90
		See also	92

### 3

<b>Key Management with KMS and CloudHSM</b>	<b>93</b>
<b>Technical requirements</b>	<b>94</b>
<b>Creating keys in KMS</b>	<b>95</b>
Getting ready	95
How to do it...	95
How it works...	99
There's more...	100
See also	101
<b>Creating keys with external key material (BYOK)</b>	<b>101</b>
Getting ready	101
How to do it...	102
How it works...	105
There's more...	105
See also	106
<b>Rotating keys in KMS</b>	<b>106</b>
Getting ready	106
How to do it...	106
How it works...	107
There's more...	108
See also	108
<b>Granting permissions programmatically with grants</b>	<b>109</b>
Getting ready	109
How to do it...	110
How it works...	112
There's more...	112
See also	114
<b>Using key policies with conditional keys</b>	<b>114</b>

Getting ready	114	There's more...	125
How to do it...	114	See also	125
How it works...	118		
There's more...	119	<b>Creating, initializing, and activating a CloudHSM cluster</b>	<b>125</b>
See also	120	Getting ready	125
<b>Sharing customer-managed keys across accounts</b>	<b>120</b>	How to do it...	125
Getting ready	120	How it works...	132
How to do it...	121	There's more...	133
How it works...	124	See also	133

## 4

### **Securing Data on S3 with Policies and Techniques** **135**

<b>Technical requirements</b>	<b>136</b>	How it works...	156
<b>Creating an S3 bucket policy</b>	<b>136</b>	There's more...	156
Getting ready	137	See also	157
How to do it...	139	<b>Protecting files with S3 versioning and object locking</b>	<b>157</b>
How it works...	143	Getting ready	157
There's more...	144	How to do it...	159
See also	146	How it works...	162
<b>Working with S3 ACLs</b>	<b>146</b>	There's more...	162
Getting ready	146	See also	164
How to do it...	148	<b>Encrypting data on S3</b>	<b>164</b>
How it works...	150	Getting ready	164
There's more...	151	How to do it...	165
See also	153	How it works...	169
<b>Creating S3 presigned URLs</b>	<b>153</b>	There's more...	169
Getting ready	153	See also	171
How to do it...	154		

## 5

### **Network and EC2 Security with VPCs** **173**

---

Technical requirements	174	See also	199
Setting up VPC plus VPC resources with minimal effort	174	Working with NACLs	199
Getting ready	175	Getting ready	199
How to do it...	175	How to do it...	199
How it works...	178	How it works...	204
There's more...	181	There's more...	204
See also	182	See also	205
Creating a bare VPC and setting up public and private subnets	182	Using a VPC gateway endpoint to connect to S3	205
Getting ready	183	Getting ready	205
How to do it...	183	How to do it...	206
How it works...	186	How it works...	207
There's more...	186	There's more...	207
See also	186	See also	208
Launching an EC2 instance with a web server using user data	187	Configuring and using VPC flow logs	208
Getting ready	187	Getting ready	208
How to do it...	187	How to do it...	208
How it works...	194	How it works...	209
There's more...	194	There's more...	210
See also	195	See also	210
Creating and configuring security groups	195	Setting up and configuring NAT gateways	210
Getting ready	195	Getting ready	210
How to do it...	195	How to do it...	211
How it works...	198	How it works...	212
There's more...	198	There's more...	213
		See also	213

## 6

**Web Security Using Certificates, CDNs, and Firewalls 215**

Technical requirements	216	How to do it...	229
Enabling HTTPS for a web server on an EC2 instance	217	How it works...	234
Getting ready	217	There's more...	235
How to do it...	217	See also	235
How it works...	219	Using a network load balancer with TLS termination at EC2	235
There's more...	219	Getting ready	235
See also	220	How to do it...	236
Creating an SSL/TLS certificate with ACM	220	How it works...	237
Getting ready	220	There's more...	237
How to do it...	221	See also	237
How it works...	223	Securing S3 using CloudFront and TLS	238
There's more...	224	Getting ready	238
See also	224	How to do it...	238
Creating ELB target groups	224	How it works...	246
Getting ready	224	There's more...	247
How to do it...	225	See also	247
How it works...	227	Using a WAF	247
There's more...	228	Getting ready	248
See also	228	How to do it...	248
Using an application load balancer with TLS termination at the ELB	228	How it works...	253
Getting ready	228	There's more...	254
		See also	254

## 7

**Monitoring with CloudWatch, CloudTrail, and Config 255**

Technical requirements	256	How it works...	257
Creating an SNS topic to send emails	256	There's more...	257
Getting ready	256	See also	257
How to do it...	256		



<b>Working with CloudWatch alarms and metrics</b>	<b>258</b>	<b>Creating a trail in CloudTrail</b>	<b>271</b>
Getting ready	258	Getting ready	271
How to do it...	258	How to do it...	271
How it works...	262	How it works...	276
There's more...	262	There's more...	277
See also	263	See also	277
<b>Creating a CloudWatch log group</b>	<b>263</b>	<b>Using Athena to query CloudTrail logs in S3</b>	<b>277</b>
Getting ready	263	Getting ready	278
How to do it...	263	How to do it...	279
How it works...	263	How it works...	280
There's more...	263	There's more...	280
See also	264	See also	281
<b>Working with EventBridge (previously CloudWatch Events)</b>	<b>264</b>	<b>Integrating CloudWatch with CloudTrail making use of a CloudFormation template</b>	<b>281</b>
Getting ready	264	Getting ready	281
How to do it...	264	How to do it...	282
How it works...	267	How it works...	284
There's more...	268	There's more...	284
See also	268	See also	284
<b>Reading and filtering logs in CloudTrail</b>	<b>268</b>	<b>Setting up and using AWS Config</b>	<b>284</b>
Getting ready	268	Getting ready	284
How to do it...	268	How to do it...	285
How it works...	270	How it works...	288
There's more...	270	There's more...	288
See also	271	See also	289

## 8

<b>Compliance with GuardDuty, Macie, Inspector, and Analyzer</b>	<b>291</b>
<b>Technical requirements</b>	<b>291</b>
<b>Setting up and using Amazon GuardDuty</b>	<b>292</b>
Getting ready	292
How to do it...	292
How it works...	295
There's more...	296
See also	298
<b>Aggregating findings from multiple accounts in GuardDuty</b>	<b>298</b>

Getting ready	298	There's more...	310
How to do it...	298	See also	311
How it works...	299		
There's more...	299	<b>Setting up and using AWS Security Hub</b>	<b>311</b>
See also	300	Getting ready	311
<b>Setting up and using Amazon Macie</b>	<b>300</b>	How to do it...	312
Getting ready	300	How it works...	317
How to do it...	301	There's more...	317
How it works...	304	See also	318
There's more...	305	<b>Using IAM Access Analyzer to inspect unused access</b>	<b>318</b>
See also	306	Getting ready	318
<b>Setting up and using Amazon Inspector</b>	<b>306</b>	How to do it...	318
Getting ready	306	How it works...	319
How to do it...	307	There's more...	320
How it works...	309	See also	321

## 9

### **Advanced Identity and Directory Management 323**

Technical requirements	324	<b>Using AWS Simple AD for creating a lightweight directory solution</b>	<b>350</b>
<b>Working with Amazon Cognito user pools</b>	<b>324</b>	Getting ready	351
Getting ready	325	How to do it...	351
How to do it...	326	How it works...	353
How it works...	336	There's more...	353
There's more...	337	See also	354
See also	338	<b>Using Microsoft Entra ID as the identity provider within AWS</b>	<b>354</b>
<b>Using identity pools to access AWS resources</b>	<b>338</b>	Getting ready	355
Getting ready	339	How to do it...	355
How to do it...	339	How it works...	363
How it works...	349	There's more...	363
There's more...	350	See also	364
See also	350		

## 10

---

### **Additional Services and Practices for AWS Security 365**

---

Technical requirements	366	How it works...	379
Setting up and using AWS RAM	366	There's more...	380
Getting ready	366	See also	380
How to do it...	367		
How it works...	368	Using security products from AWS	
There's more...	368	Marketplace	380
See also	368	Getting ready...	380
		How to do it...	380
Storing sensitive data with the		How it works...	382
Systems Manager Parameter Store	369	There's more...	382
Getting ready	369	See also	382
How to do it...	369	Using AWS Trusted Advisor for	
How it works...	371	recommendations	382
There's more...	372	Getting ready	382
See also	372	How to do it...	383
		How it works...	384
Using AWS Secrets Manager to		There's more...	384
manage RDS credentials	372	See also	384
Getting ready	373		
How to do it...	374	Using AWS Artifact for compliance	
How it works...	378	reports	384
There's more...	378	Getting ready	384
See also	378	How to do it...	385
		How it works...	386
Creating an AMI instead of using		There's more...	387
EC2 user data	379	See also	387
Getting ready	379		
How to do it...	379		

---

### **Index 389**

---

### **Other Books You May Enjoy 402**

---

# Preface

The second edition of *AWS Security Cookbook* discusses practical solutions to the most common problems faced by security consultants while securing their infrastructure. This book explores various AWS services and features that help implement security models and concepts such as the **Confidentiality, Integrity, and Availability (CIA)** triad, the **Authentication, Authorization, and Accounting (AAA)** triad, and non-repudiation. The book begins by getting you familiar with essential AWS security features such as **Identity and Access Management (IAM)**, account aliases, and billing alerts, then delves deeper into access management, key management, data security, network security, web security, monitoring, compliance, advanced identity management, and additional security services and practices. Throughout the book, you will come across AWS Security services such as IAM, AWS Organizations, IAM Identity Center, **Key Management Service (KMS)**, CloudHSM, S3, **Virtual Private Cloud (VPC)**, CloudWatch, CloudTrail, Config, GuardDuty, Macie, Inspector, Security Hub, Cognito, Resource Access Manager, Systems Manager Parameter Store, Secrets Manager, Trusted Advisor, and AWS Artifact. Each chapter focuses on essential security areas and progresses toward cloud security best practices and integrating additional security services. By the end of this book, you will be adept with all of the techniques pertaining to securing AWS deployments along with having help to prepare for the AWS Certified Security – Specialty certification.

## Who this book is for

If you are an IT security professional, cloud security architect, or cloud application developer working on security-related roles and are interested in using the AWS infrastructure for secure application deployment, then this book is for you. This book will also benefit individuals interested in taking up the AWS Certified Security – Specialty certification.

## What this book covers

*Chapter 1, Setting up AWS Accounts and Organization*, introduces essential AWS security features, beginning with setting up the IAM service, account aliases, and billing alerts. It also covers multi-account management with AWS Organizations and user management with IAM Identity Center for centralized identity creation and access management. Completing this chapter prepares you for the enterprise-level AWS security management concepts discussed within this book.

*Chapter 2, Access Management with IAM Policies and Roles*, shows how secure access management in AWS is vital for controlling who can access resources and what actions they can perform, ensuring the environment remains secure and compliant. This chapter covers IAM policies, IAM roles, and various policy types, providing detailed recipes for creating and managing policies, setting permission boundaries, and implementing cross-account and cross-service access. This knowledge is essential for establishing a robust and scalable security posture across AWS infrastructure.

*Chapter 3, Key Management with KMS and CloudHSM*, covers how encryption converts plain text into unreadable cipher text to protect messages from unauthorized access, and decryption reverses this process. Using AWS **KMS** and CloudHSM, this chapter covers creating, managing, and securing encryption keys, supporting both symmetric and asymmetric encryption. It includes recipes for key creation, key rotation, permissions management, and cross-account key sharing to ensure robust data protection in AWS.

*Chapter 4, Securing Data on S3 with Policies and Techniques*, focuses on securing Amazon S3, AWS's object storage service, which uses a unique key-value system for storing data. Building on IAM policies from *Chapter 2*, we'll explore securing S3 data with **Access Control Lists (ACLs)**, bucket policies, pre-signed URLs, encryption, versioning, and replication. Recipes include creating bucket policies, working with ACLs, generating pre-signed URLs, protecting files with versioning, and encrypting S3 data.

*Chapter 5, Network and EC2 Security with VPCs*, covers Amazon VPC, a key AWS component that enables the creation of private, isolated networks within the AWS cloud. Users have full control over their virtual network, allowing customization of IP ranges, subnets, route tables, and gateways. The chapter includes recipes for setting up VPC resources, creating subnets, launching EC2 instances, configuring security groups and NACLs, connecting to S3 via gateway endpoints, using VPC flow logs, and setting up NAT gateways.

*Chapter 6, Web Security Using Certificates, CDNs, and Firewalls*, emphasizes web security through certificates, **Content Delivery Networks (CDNs)**, and firewalls. It covers using X.509 certificates for TLS encryption, leveraging load balancers for efficient traffic distribution and TLS termination, and utilizing CDNs to enhance load times and security. Additionally, it explores firewall integration within AWS to define and enforce network security perimeters. Practical implementations of these components provide the knowledge to build a secure, scalable web infrastructure.

*Chapter 7, Monitoring with CloudWatch, CloudTrail, and Config*, explores the role of monitoring and auditing in security through Amazon CloudWatch, AWS CloudTrail, and AWS Config. CloudWatch handles logging, monitoring, and alerting; CloudTrail logs AWS API calls; and AWS Config evaluates configurations against rules. The chapter also covers using **Simple Notification Service (SNS)** for notifications and includes practical recipes for setting up these services and querying logs.

*Chapter 8, Compliance with GuardDuty, Macie, Inspector, and Analyzer*, details how regular compliance checks and notifications are crucial for maintaining account security. This chapter introduces AWS services such as Amazon GuardDuty, Amazon Macie, and Amazon Inspector, which leverage machine learning and advanced algorithms for compliance monitoring. It covers practical steps for setting up and using these services, along with AWS Security Hub and IAM Analyzer for comprehensive security management.

*Chapter 9, Advanced Identity and Directory Management*, builds on previous chapters' IAM foundations. It explores advanced user identity and directory management with Amazon Cognito and AWS directory services. It covers Cognito's user pools for user registration and authentication, identity pools for AWS service access, and directory solutions such as AWS Simple AD and integration with Microsoft Entra ID. The chapter includes practical recipes, but some may require additional knowledge of Microsoft products and AWS services.

*Chapter 10, Additional Services and Practices for AWS Security*, explores additional AWS security services and practices to further secure your AWS infrastructure. It introduces AWS Resource Access Manager, Systems Manager Parameter Store, Secrets Manager, Trusted Advisor, and AWS Artifact, along with the use of **Amazon Machine Images (AMIs)** and AWS Marketplace security products. Practical recipes are provided for each service, with further learning resources linked for deeper exploration.

## To get the most out of this book

You will need a working AWS account to practice the recipes within this book. You should already have some basic knowledge of AWS services such as IAM, S3, EC2, and VPC. Moreover, basic knowledge of cloud computing, computer networking, and IT security concepts can help you grasp the contents of this book faster.

**If you are using the digital version of this book, we advise you to type the code yourself or access the code via the GitHub repository (link available in the next section). Doing so will help you avoid any potential errors related to the copying and pasting of code.**

## Download the example code files

You can download the example code files for this book from GitHub at <https://github.com/PacktPublishing/AWS-Security-Cookbook-Second-Edition>. If there's an update to the code, it will be updated in the GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at <https://github.com/PacktPublishing/>. Check them out!

## Code in Action

The *Code in Action* videos for this book can be viewed at <http://bit.ly/2OQfDum>.

## Conventions used

There are a number of text conventions used throughout this book.

**Code in text:** Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: “Create a bucket policy to allow our `awssecb_user1` user to access it and save it as `bucket-policy-allow-list.json`.”

A block of code is set as follows:

```
"Principal": {
  "AWS": [
    "arn:aws:iam::873506153865:root",
    "arn:aws:iam::671100771477:root"
  ]
}
```

Any command-line input or output is written as follows:

```
aws sts assume-role --role-arn ROLE_ARN --role-session-name SESSION_
NAME --profile AWSSecCBUser1S --external-id awssecb-cust-0819
```

**Bold:** Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: “For **NAT gateways** (\$), select **1 in 1 AZ**, for **VPC endpoints**, select **S3 Gateway**, and for **DNS options**, select **Enable DNS hostnames** and **Enable DNS resolution**.”

### Tips or important notes

Appear like this.

## Sections

In this book, you will find several headings that appear frequently (*Getting ready*, *How to do it...*, *How it works...*, *There's more...*, and *See also*).

To give clear instructions on how to complete a recipe, use these sections as follows:

### Getting ready

This section tells you what to expect in the recipe and describes how to set up any software or any preliminary settings required for the recipe.

---

## How to do it...

This section contains the steps required to follow the recipe.

## How it works...

This section usually consists of a detailed explanation of what happened in the previous section.

## There's more...

This section consists of additional information about the recipe in order to make you more knowledgeable about the recipe.

## See also

This section provides helpful links to other useful information for the recipe.

## Get in touch

Feedback from our readers is always welcome.

**General feedback:** If you have questions about any aspect of this book, email us at [customercare@packtpub.com](mailto:customercare@packtpub.com) and mention the book title in the subject of your message.

**Errata:** Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit [www.packtpub.com/support/errata](http://www.packtpub.com/support/errata), select your book, click on the Errata Submission Form link, and enter the details.

**Piracy:** If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at [copyright@packtpub.com](mailto:copyright@packtpub.com) with a link to the material.

**If you are interested in becoming an author:** If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit [authors.packtpub.com](http://authors.packtpub.com).



## Share Your Thoughts

Once you've read *AWS Security Cookbook*, we'd love to hear your thoughts! Please [click here](#) to go straight to the Amazon review page for this book and share your feedback.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

---

## Download a free PDF copy of this book

Thanks for purchasing this book!

Do you like to read on the go but are unable to carry your print books everywhere?

Is your eBook purchase not compatible with the device of your choice?

Don't worry, now with every Packt book you get a DRM-free PDF version of that book at no cost.

Read anywhere, any place, on any device. Search, copy, and paste code from your favorite technical books directly into your application.

The perks don't stop there, you can get exclusive access to discounts, newsletters, and great free content in your inbox daily

Follow these simple steps to get the benefits:

1. Scan the QR code or visit the link below



<https://packt.link/free-ebook/9781835081891>

2. Submit your proof of purchase
3. That's it! We'll send your free PDF and other benefits to your email directly



# 1

## Setting Up AWS Accounts and Organization

An application or platform's security is often characterized by features such as confidentiality, integrity, availability, authentication, authorization, accounting, and non-repudiation. These features are grouped into the **Confidentiality, Integrity, and Availability (CIA)** triad and the **Authentication, Authorization, and Accounting (AAA)** triad. A solid grasp of these security features will facilitate better understanding and implementation of the AWS security concepts detailed in this book.

In this chapter, we will first learn about setting up the **Identity and Access Management (IAM)** service for a new AWS account along with **account aliases** and **billing alerts**. Then, we will learn to set up the **AWS Organizations** service that allows us to create and manage multiple AWS accounts from within a single **management account**. We will also learn about user management and **Single Sign-On (SSO)** using **AWS IAM Identity Center** (formerly known as AWS SSO), which centralizes identity creation and access management across AWS accounts and apps and is recommended for organizations of all sizes and types.

This chapter is slightly longer than the rest of the chapters in this book since it sets the stage for other chapters. We could skip the second and third recipes within this chapter regarding setting up AWS Organizations and IAM Identity Center and execute most of the recipes in other chapters on a standalone AWS account. However, if our goal is to work in an enterprise environment, it would be good to complete all the recipes within this chapter before proceeding with the rest of the book.

This chapter includes the following recipes:

- Setting up IAM, account aliases, and billing alerts
- Multi-account management with AWS Organizations
- User management and SSO with IAM Identity Center

## Technical requirements

Before diving into the recipes of this chapter, we need to ensure we have the following knowledge and requirements in place:

- **AWS account:** It is recommended to use a new AWS account for this chapter. We can sign up for a free tier account at <https://aws.amazon.com>.
- **Permissions:** We need to operate as the root user or possess administrative permissions to configure IAM, AWS Organizations, and IAM Identity Center.
- **Familiarity with AWS:** A working knowledge of the AWS Management Console and AWS services such as IAM and S3 will benefit us.
- **Internet connection:** Due to AWS's cloud-based nature, a stable internet connection is essential for accessing and managing AWS services.
- **AWS Command Line Interface (CLI):** To execute AWS CLI commands, we need AWS CLI V2. Two recommended methods for working with AWS CLI V2 are using the AWS IAM Identity Center to set it up on a local or virtual machine or using **AWS CloudShell**. We will learn how to set up AWS CLI V2 using IAM Identity Center in the *User management and SSO with IAM Identity Center* recipe.

### Important note

We can also configure the AWS CLI using the traditional access keys. However, both IAM Identity Center and CloudShell utilize short-term credentials that are session-based, thus reducing the risks associated with the traditional long-lived access keys. AWS CloudShell offers the added convenience of a browser-based shell integrated directly into the AWS Management Console, facilitating on-the-fly operations without the need for local setup. However, CloudShell may not be available in all regions.

The code files for this book are available at <https://github.com/PacktPublishing/AWS-Security-Cookbook-Second-Edition>. The code files for this chapter are available at <https://github.com/PacktPublishing/AWS-Security-Cookbook-Second-Edition/tree/main/Chapter01>.

---

## Setting up IAM, account aliases, and billing alerts

IAM is the primary service in AWS that is used to manage access to AWS resources. After setting up an AWS account, it is recommended to perform some basic IAM configurations to enhance the security of our account such as securing the **root account** with **multi-factor authentication (MFA)**. MFA is a security mechanism that requires users to provide two or more verification factors to gain access to a resource, such as a username and password, and a code sent to a phone. IAM offers a checklist to guide these initial activities.

While not part of the AWS checklist, it is also recommended to set up an account alias and create a billing alarm in a new account. An account alias is a unique identifier that we can create for our AWS account instead of the default 12-digit account ID in our account's sign-in URL. Creating an account alias enhances the user experience by providing a personalized and memorable sign-in URL, increases security by obscuring the actual account number, and allows for branding consistency by incorporating our organization's name into the AWS login process.

Setting up a billing alarm in a new AWS account offers the key advantage of monitoring and managing costs effectively. It alerts us when our account spending exceeds a predefined threshold, thus helping to prevent unexpected charges and maintain budget control.

### Getting ready

We need a newly created AWS account to complete all the steps in this recipe. If the account is not new, we can verify whether everything has been configured correctly by following this recipe and configuring anything that is missing.

To set up MFA using a virtual MFA device within this recipe, we need to install an authenticator app on our mobile, and **Google Authenticator** is a popular authenticator app that we can use. We can also use a **YubiKey Universal 2nd Factor (U2F) security key**, any **U2F-compliant device**, or a **hardware MFA device**. U2F is an authentication standard for securely accessing online services with only a security key and without any drivers or client software.

### How to do it...

First, we will set up IAM for a new AWS account. Then we will set up an account alias for a better user experience and a billing alarm to prevent unplanned usage and billing.

## Setting up IAM for AWS accounts

We can set up IAM for a new AWS account as follows:

1. Log in to the AWS account using the root user email credential and follow these steps from within the IAM dashboard. For a new AWS account, the IAM dashboard should look as follows:

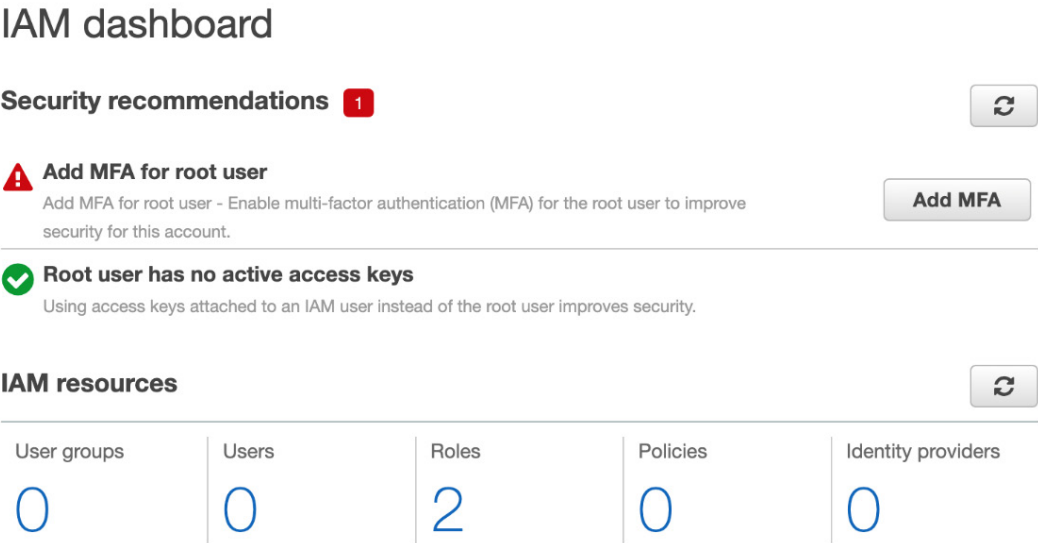


Figure 1.1 – The IAM dashboard for a new account

2. Click on **Add MFA** under **Security recommendations**.
3. If you are not redirected to the **Assign MFA** tab, go to the **Assign MFA** tab manually.
4. Enter a meaningful value for the MFA device name that can help us recognize the device, then select **Authenticator app** as shown in the following figure. If we set up a different option, as discussed in the *Getting ready* section, we can select it here instead of the **Authenticator app** option.

Step 1 of 2

## Select MFA device

### Specify MFA device name


Device name  
Enter a meaningful name to identify this device.


HKAuthApp

Maximum 128 characters. Use alphanumeric and '+', '=', '@', '-', '\_' characters.

### Select MFA device [Info](#)

Select an MFA device to use, in addition to your username and password, whenever you need to authenticate.

☒  **Authenticator app**  
Authenticate using a code generated by an app installed on your mobile device or computer.

☐  **Security Key**  
Authenticate using a code generated by touching a YubiKey or other supported FIDO security key.


☐  **Hardware TOTP token**  
Authenticate using a code displayed on a hardware Time-based one-time password (TOTP) token.

Figure 1.2 – Selecting an MFA device

5. Scroll down and click **Next**. AWS will now provide a QR code.

#### Important note

We can save the QR code image in a secure place if we ever want to reconfigure the authenticator app without accessing the current authenticator app setup, for example, if our current mobile stops working. Alternatively, we can contact AWS support in case of such an event and they can help us reset the authenticator app configuration.

6. Scan the QR code using an authenticator app (e.g., Google Authenticator) installed on your mobile device and enter two successful token keys to activate it.

After MFA has been activated, we will need to provide a token from this app, along with a username and password, to log in to the AWS console.



7. Go back to the IAM dashboard and ensure that all the checkmarks within the security recommendations we saw in *Figure 1.1* are green now.

## IAM dashboard

### Security recommendations

- ✓ **Root user has MFA**  
Having multi-factor authentication (MFA) for the root user improves security for this account.
- ✓ **Root user has no active access keys**  
Using access keys attached to an IAM user instead of the root user improves security.

### IAM resources

User groups	Users	Roles	Policies
0	0	2	0

Figure 1.3 – Security recommendations on the IAM dashboard

Once all security recommendations are green, indicating compliance, AWS may not display these recommendations again.

Next, we will set up an account alias and billing alerts. It is recommended to do this on a new AWS account even if these things are not part of the security recommendations.

### Setting up an account alias

We can configure an account alias within the IAM dashboard as follows:

1. Within the IAM page, under **Account Alias**, click **Create** to open the **Create alias** pop-up page.

**AWS Account**

Account ID  
370598287390

Account Alias  
[Create](#)

Sign-in URL for IAM users in this account  
<https://370598287390.signin.aws.amazon.com/console>

Figure 1.4 – The AWS account details

2. In the **Create alias** pop-up page, type in a unique and meaningful alias for our account under **Preferred alias** and click on **Create alias**.

This will create an account alias for our account. The account alias will replace the account ID from the URL under **Sign-in URL for IAM users in this account** as shown in *Figure 1.4* and make it easier for our IAM users to remember the sign-in URL. Please note that IAM users will still be able to log in using the default sign-in URL with the account ID that we saw in *Figure 1.4*.

Now, let us also create a billing alarm.

### ***Creating a billing alarm***

In this section, we will set up a billing alarm to let us know when we exceed a set limit:

1. Log in to the AWS management console and from the drop-down menu next to the account name in the upper-right corner of the screen, click on **Billing and Cost Management**.

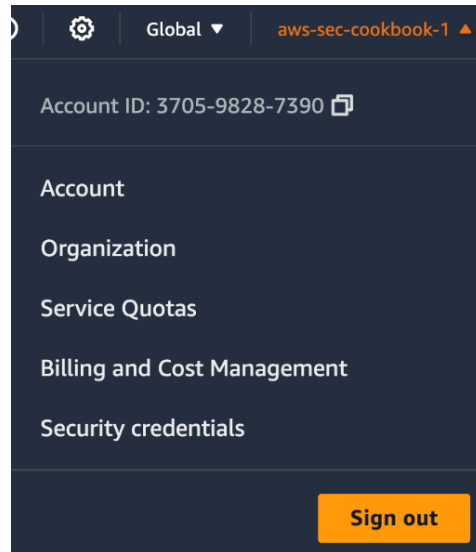


Figure 1.5 – The account drop-down menu

2. On the page for **Billing and Cost Management home**, click on **Billing preferences** from the left sidebar.
3. Click on **Edit** within the **Alert preferences** pane.

4. On the **Alert Preferences** page, select the **Receive CloudWatch billing alerts** checkbox. If you are using a free tier account, you can also select **Receive AWS Free Tier alerts**, and optionally give an additional email address to receive alerts in the **Additional email address to receive alerts – optional** textbox. Click **Update** to save your preferences.
5. Go to the CloudWatch service dashboard, set the region to **US East (N. Virginia)**, expand **Alarms** on the left, and click on **All alarms**. At the time of writing this book, AWS only allows us to create a billing alarm if our region is set to **US East (N. Virginia)**. We will learn more about the CloudWatch service in a later chapter of this book.
6. Click on **Create alarm**, and on the **Create alarm** page, click on **Select metric**.
7. In the **Browse** tab, click **Billing**, and then click **Total Estimated Charge**.
8. Select the checkbox for the **EstimatedCharges** metric as shown in the following figure, and then click **Select metric**.

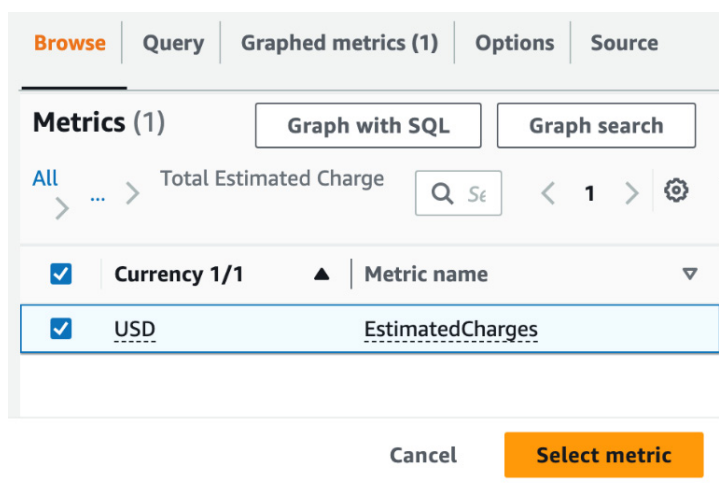


Figure 1.6 – Configuring metrics

9. Leave the value for **Metric name** as **EstimatedCharges** and **Currency** as **USD**. For **Statistic**, choose **Maximum**, and for **Period**, choose **6 hours**.

10. Under **Conditions**, for **Threshold type**, choose **Static**, for **Whenever EstimatedCharges is ...**, choose **Greater**, and for **than...**, define the value for triggering the alarm. Also, expand **Additional configuration**. For **Datapoints to alarm**, specify 1 out of 1, and for **Missing data treatment**, select **Treat missing data as missing**.

### Conditions

#### Threshold type

☒ **Static**

Use a value as a threshold

☐ **Anomaly detection**

Use a band as a threshold

#### Whenever EstimatedCharges is...

Define the alarm condition.

☒ **Greater**

> threshold

☐ **Greater/Equal**

>= threshold

☐ **Lower/Equal**

<= threshold

☐ **Lower**

< threshold

#### than...

Define the threshold value.

10

USD

Must be a number

### ▼ Additional configuration

#### Datapoints to alarm

Define the number of datapoints within the evaluation period that must be breaching to cause the alarm to go to ALARM state.

1

out of

1

#### Missing data treatment

How to treat missing data when evaluating the alarm.

Treat missing data as missing



Cancel

Next

Figure 1.7 – Configuring the conditions for the metric - Conditions

11. Click **Next** to go to the **Notification** page. In the **Notification** page, select **In alarm**, then select **Create new topic**, provide a name for the topic and an email to receive communications as shown in the following figure, and click **Create topic**.

## Notification

Alarm state trigger

Define the alarm state that will trigger this action.

☒ **In alarm**  
The metric or expression is outside of the defined threshold.

☐ **OK**  
The metric or expression is within the defined threshold.

☐ **Insufficient data**  
The alarm has just started or not enough data is available.

Remove

Send a notification to the following SNS topic

Define the SNS (Simple Notification Service) topic that will receive the notification.

☐ Select an existing SNS topic

☒ **Create new topic**

☐ Use topic ARN to notify other accounts

Create a new topic...

The topic name must be unique.

Default\_CloudWatch\_Alarms\_Topic

SNS topic names can contain only alphanumeric characters, hyphens (-) and underscores (\_).

Email endpoints that will receive the notification...

Add a comma-separated list of email addresses. Each address will be added as a subscription to the topic above.

team@cloudericks.com

user1@example.com, user2@example.com

Create topic

Add notification

Figure 1.8 – Configuring notifications for the metric - Notification

We created a new **Simple Notification Service (SNS)** topic to send emails. We may also select an existing SNS topic instead of creating a new one.

12. Click **Next**, and under **Name and description**, enter a name for the alarm.
13. Click **Next** to proceed to the **Preview and create** page. Click **Create alarm**. The alarm should now be created successfully.

Please note that Amazon SNS doesn't send messages to an endpoint until the subscription to the topic is confirmed.

## How it works...

IAM is the AWS service that helps us manage and verify the identity of users within AWS (authentication) and verify their permissions to AWS services (authorization). IAM is a global service and not tied to a region. IAM has four core concepts:

- **Users:** A user can be created in IAM and given the necessary permissions to access AWS resources.
- **Groups:** Users can be added to groups. Permission can then be given to groups instead of individual users. This is a recommended best practice.
- **Policies:** Policies are JSON documents that define the permissions for users or groups.
- **Roles:** Roles are generally used for giving users temporary permissions to access an AWS service. For example, we can attach a role with S3 permissions to an EC2 service. Roles are also used for switching roles, as we will see in *Chapter 2*.

The root user account is the account that we log in to using the primary email. It has access to everything in our account. The IAM dashboard provides a set of checklist items to keep our root account secure. The first item in the IAM checklist checks whether we have enabled MFA for our root account. MFA will enforce an additional level of authentication, apart from the username and password, using tokens from a virtual or hardware MFA device.

The second item in the checklist checks whether we have active access keys for our root account that can be used for programmatic access. It is good practice to use the root for creating other accounts and making necessary configurations, and then use those accounts for our day-to-day activities. As we will see in further recipes within this chapter, we can use the IAM Identity Center along with the AWS Organizations service to better manage user identities across AWS accounts in an organization.

We also set up an account alias and a billing alarm even though they were not part of the IAM security recommendations checklist. As we saw in the recipe introduction, creating an AWS account alias enhances the user experience, increases security, and allows for branding consistency. A billing alarm will trigger an alarm and let us know when we exceed the set limit. It is a good practice to always set a billing alarm to avoid accidental usage and unplanned expenses.

## There's more...

Let us also quickly go through some important concepts related to IAM and security:

- **Authentication** is the process of verifying a user's identity with a username and password, or credentials such as the access key and the secret access key. There are primarily two types of access credentials in AWS for authenticating users:
  - **Access key ID and secret access key:** These are used for programmatic access, and are used with AWS APIs, CLI, SDK, and any development tools.
  - **Username and password:** These are used for managing console access.
- **Authorization** is the process of checking whether a user has the right permissions to perform an action and is usually defined using policies. We will learn more about IAM policies in *Chapter 2*.
- **Confidentiality** is making sure that the data that's sent from the source is not read by anyone else during transit. This can be made possible using cryptography.
- **Data integrity** is making sure the data has come from the right person and has not been tampered in between. This is possible using cryptography.
- **Availability** makes sure that the service can be used when it is needed.
- **Accounting** helps us identify the responsible parties in case of a security event.
- **Non-repudiation** prevents a user from denying an activity. Cryptography comes to our aid here as well.
- The **AWS shared responsibility model** defines the responsibilities of AWS and its customers in securing our solutions in the AWS cloud. AWS is responsible for the security of the cloud, which involves safeguarding the infrastructure that powers all services provided in the AWS cloud. This includes the hardware, software, networking, and facilities that operate AWS cloud services. On the other hand, customers are accountable for security in the cloud. This responsibility encompasses the management of workloads deployed in the cloud, the guest operating systems used, and the configurations of network, host, IAM, and storage resources for data management and business communication. It also involves the regular updating and patching of software on cloud resources based on the cloud abstraction (e.g., infrastructure as a service) used.
- Third-party auditors evaluate AWS IAM regularly for its compliance with a range of standards, including the **Service Organization Control (SOC)**, **Payment Card Industry Data Security Standard (PCI DSS)**, **Federal Risk and Authorization Management Program (FedRAMP)**, and **International Organization for Standardization (ISO)**, among other standards.

## More about IAM users and groups

In this recipe, we did not create IAM users or user groups because it is now recommended to use IAM Identity Center users instead of IAM users. However, if needed, we can create users and user groups from the left sidebar of the IAM dashboard.

The primary concern with IAM users is that they are associated with long-term credentials such as access keys, which can pose security risks if they are not managed properly. Here are some common use cases for AWS access keys as per AWS, along with recommendations for safer alternatives:

Alternative	Use case	Recommendations
<b>CLI</b>	Accessing the AWS account via the AWS CLI.	In this case do the following: <ul style="list-style-type: none"> <li>Use <b>AWS CloudShell</b>, a browser-integrated CLI, for executing commands.</li> <li>Opt for the AWS CLI V2 and set up authentication via a user in the <b>IAM Identity Center</b>; we will delve into the IAM Identity Center later in this.</li> </ul>
<b>Local code</b>	Accessing the AWS account from local development environments.	Employ an <b>Integrated Development Environment (IDE)</b> equipped with the AWS Toolkit, which facilitates authentication through the IAM Identity Center.
<b>Application running on an AWS compute service</b>	Managing application code on AWS compute services such as Amazon EC2, Amazon ECS, or AWS Lambda.	Assign an IAM role to compute resources such as EC2 instances or Lambda functions, ensuring the automatic provision of temporary credentials for access.
<b>Third-party Service</b>	This is used for enabling third-party applications or services that oversee or interact with AWS resources.	As a standard, opt for temporary security credentials through IAM roles, avoiding the creation of enduring credentials such as access keys. Refrain from generating AWS account root user access keys.
<b>Applications running outside AWS</b>	Managing application code on AWS compute services such as Amazon EC2, Amazon ECS, or AWS Lambda.	While it's acceptable to employ an access key for this use case, ensure that you do the following: <ul style="list-style-type: none"> <li>Avoid storing access keys in plaintext, within code repositories, or directly in code.</li> <li>Deactivate or remove access keys that are redundant.</li> <li>Implement least-privilege permissions.</li> <li>Regularly rotate access keys.</li> </ul>

Table 1.1 – Use cases for AWS access keys as per AWS



The preceding recommendations aim to reduce the potential for security breaches.

## See also

- We can learn more about IAM and its security tools and features such as the IAM analyzer, IAM access advisor, and IAM credentials report at <https://www.cloudericks.com/blog/getting-started-with-aws-iam>.
- We learned about billing alarms in this recipe. We can learn about AWS Budgets and understand its differences from billing alarms at <https://www.cloudericks.com/blog/getting-started-with-aws-budgets>.
- We briefly touched on Amazon SNS within this recipe. To learn more about SNS, we can refer to <https://www.cloudericks.com/blog/getting-started-with-amazon-sns-service>.
- This book does not cover AWS basics beyond the security domain. We can read about cloud computing and AWS basics at <https://www.cloudericks.com/blog/getting-started-with-aws-cloud>.
- Read more about the AWS shared responsibility model at <https://aws.amazon.com/compliance/shared-responsibility-model>.
- Read more about compliance validation for AWS IAM at <https://docs.aws.amazon.com/IAM/latest/UserGuide/iam-compliance-validation.html>.

## Multi-account management with AWS Organizations

Organizations generally have multiple AWS accounts categorized based on usages such as production, development, testing, and so on. The AWS Organizations service helps us centrally manage all our AWS accounts, and its **Organizational Units (OUs)** feature helps us maintain the AWS accounts in usage-based hierarchies. In this recipe, we will learn how to create AWS Organization and OUs, as well as add AWS accounts to OUs. We will create the AWS Organization from the AWS Management Console but will create OUs and add accounts both from the Management Console and the CLI.

### Getting ready

We will need a working AWS account that is not part of an AWS Organization to complete all the steps within this recipe.

For parts of the recipe where we use CLI commands, we need AWS CLI V2 configured as discussed within the *Technical requirements* section.

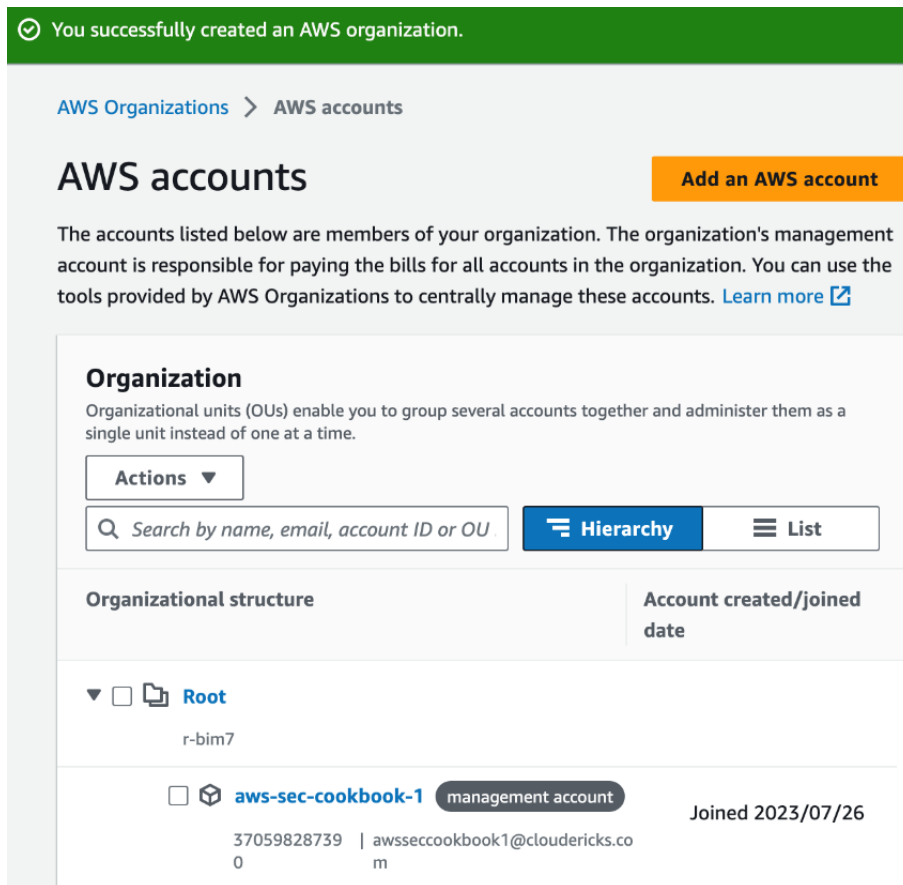
## How to do it...

We will first create an AWS Organization and will then create OUs and AWS accounts within that AWS Organization.

### *Creating an AWS organization from the management console*

Let us first create an AWS Organization as follows:

1. Log in to the AWS Management Console as a root user or a user with administrator permissions and go to the **AWS Organizations** service dashboard.
2. Click on **Create an organization**. It will create an organization and forward us to the **AWS accounts** page, which should look like the following:



✓ You successfully created an AWS organization.

[AWS Organizations](#) > [AWS accounts](#)

## AWS accounts

[Add an AWS account](#)

The accounts listed below are members of your organization. The organization's management account is responsible for paying the bills for all accounts in the organization. You can use the tools provided by AWS Organizations to centrally manage these accounts. [Learn more](#)

### Organization

Organizational units (OUs) enable you to group several accounts together and administer them as a single unit instead of one at a time.

Actions ▼

Search by name, email, account ID or OU

[Hierarchy](#) [List](#)

Organizational structure	Account created/joined date
▼ <input type="checkbox"/> <b>Root</b> r-bim7	
<input type="checkbox"/> <b>aws-sec-cookbook-1</b> <span>management account</span> 37059828739   awsseccookbook1@cloudericks.co 0 m	Joined 2023/07/26

Figure 1.9 – Accounts in AWS Organizations

If we check the left sidebar on the same page, we can see **Organization ID** as shown in the following figure:

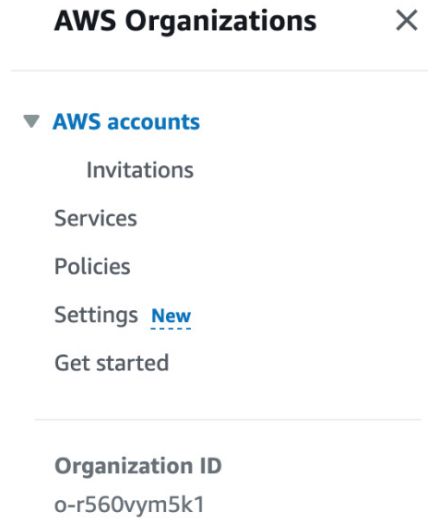


Figure 1.10 – The AWS Organizations sidebar

Next, we will create an OU under the root OU.

### *Creating an OU and account from the Management Console*

To create an OU under the root OU, we can proceed with the following steps:

1. Go to the **AWS accounts** page on the **AWS Organizations** service dashboard.
2. Select the **Root** OU, and from the **Actions** menu, select **Create new** under **Organizational unit**.

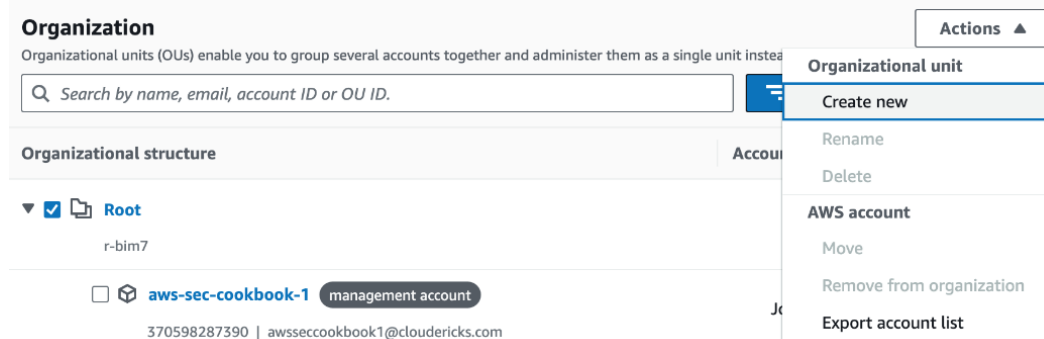


Figure 1.11 – Creating a new organizational unit

3. On the next screen, enter Sandbox under **Organizational unit name** and click on **Create organizational unit**.

We can now create an AWS account and move it under the **Sandbox** OU as follows:

1. Go to the **AWS accounts** page on the **AWS Organizations** service dashboard.
2. Click on **Add an AWS account**.
3. Select **Create an AWS Account** and provide the `awsseccb-sandbox-1` value for **AWS account name**. For **Email address of the account's owner**, provide an email address you have access to. Enter the `OrganizationAccountAccessRole` value for **IAM role name**.

AWS Organizations > AWS accounts > Add an AWS account

## Add an AWS account

You can add an AWS account to your organization either by creating an account or by inviting one or more existing AWS accounts to join your organization.

☒ **Create an AWS account**  
Create an AWS account that is added to your organization.

☐ **Invite an existing AWS account**  
Send an email request to the owner of the account. If they accept, the account joins the organization.

### Create an AWS account

**AWS account name**

**Email address of the account's owner**

**IAM role name**  
The management account can use this IAM role to access resources in the member account.

Figure 1.12 – Adding an account to an organization

4. Scroll down and click on **Create AWS account**.

We should immediately see a screen with a message that says **AWS is creating 1 account**. It could take some time for the account to be created.

- Once the account is created, from the **AWS Accounts** page within the Organization, select the newly created account. From the **Actions** drop-down menu, select **Move**.

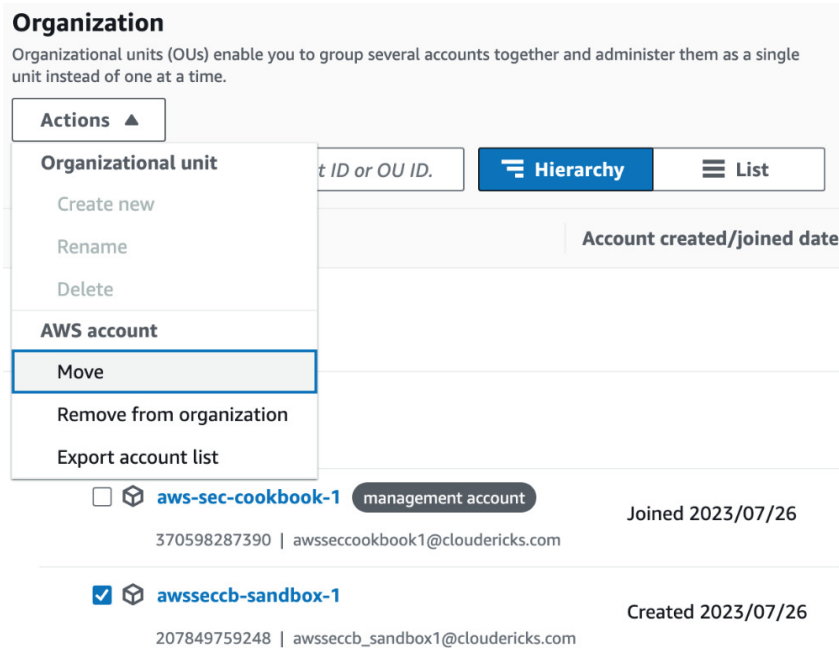


Figure 1.13 – Selecting an account and moving between OUs

- Select the desired OU in the **Destination** section and click **Move AWS account**.

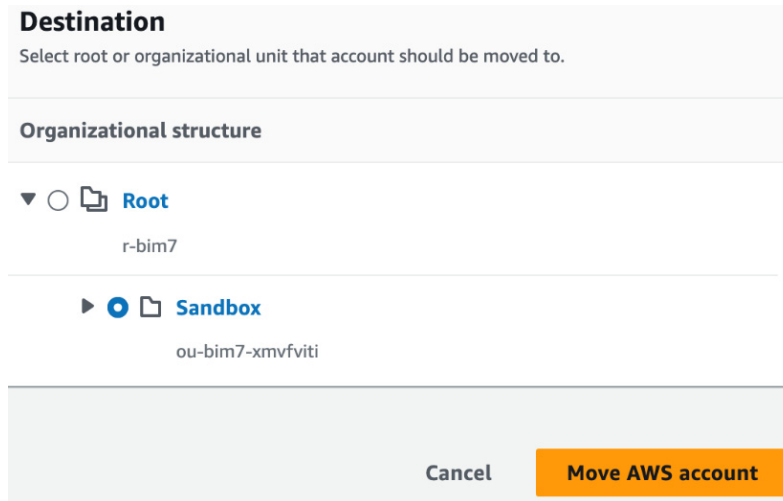


Figure 1.14 – Selecting a destination OU for moving accounts

The newly created account should be now part of the selected OU.

### *Creating an OU and account from the CLI*

In this section, we will create an OU and account from the CLI. Please remember to replace my IDs with your own IDs from the relevant previous steps while executing the CLI commands. Commands are also provided with the code files. Also, if we are executing the commands from the AWS CloudShell, we do not have to specify the CLI profile. Let us get started:

1. Create an OU called Workloads under Root OU using the `create-organizational-unit` subcommand using the ID of our Root OU:

```
aws organizations create-organizational-unit --parent-id r-bim7
--name Workloads --profile awssecadmin
```

This should give us a response similar to the following:

```
{
  "OrganizationalUnit": {
    "Id": "ou-bim7-0s1nqy2w",
    "Arn": "arn:aws:organizations::370598287390:
ou/o-r560vym5k1/ou-bim7-0s1nqy2w",
    "Name": "Workloads"
  }
}
```

Figure 1.15 – The response for the `create-organizational-unit` subcommand

2. We can create an AWS account using the `create-account` subcommand:

```
aws organizations create-account --email awsseccb_sandbox2@
cloudericks.com --account-name awsseccb-sandbox-2 --profile
awssecadmin
```

This should give us a response similar to the following:

```
{
  "CreateAccountStatus": {
    "Id": "car-6582b2c63be845ebaa474c9268cea8c1",
    "AccountName": "awsseccb-sandbox-2",
    "State": "IN_PROGRESS",
    "RequestedTimestamp": "2023-08-02T11:20:03.813000+05:30"
  }
}
```

Figure 1.16 – The response for the `create-account` subcommand

3. We can check the status of our request using the `describe-create-account-status` subcommand by providing the request ID we received in the previous step:

```
aws organizations describe-create-account-status --create-
account-request-id car-6582b2c63be845ebaa474c9268cea8c1
--profile awssecadmin
```

If the request succeeds, we should get the following response:

```
{
  "CreateAccountStatus": {
    "Id": "car-6582b2c63be845ebaa474c9268cea8c1",
    "AccountName": "awsseccb-sandbox-2",
    "State": "SUCCEEDED",
    "RequestedTimestamp": "2023-08-02T11:20:03.864000+05:30",
    "CompletedTimestamp": "2023-08-02T11:20:06.987000+05:30",
    "AccountId": "206722961012"
  }
}
```

Figure 1.17 – The response for `describe-create-account-status`

4. We can verify that the account was created under the root OU and get the root's ID using the `list-parents` subcommand by providing the account ID we received in the previous step:

```
aws organizations list-parents --child-id 206722961012 --profile
awssecadmin
```

This should give us a response similar to the following:

```
{
  "Parents": [
    {
      "Id": "r-bim7",
      "Type": "ROOT"
    }
  ]
}
```

Figure 1.18 – The response for the `list-parents` subcommand

5. Move our new account from the root OU into the new OU we created earlier from CLI by providing the account ID, root ID, and OU ID from the previous steps using the `move-account` subcommand:

```
aws organizations move-account --account-id 206722961012
--source-parent-id r-bim7 --destination-parent-id ou-bim7-
0s1nqy2w --profile awssecadmin
```

This command does not return anything.

6. Check the parent for our account using the `list-parents` subcommand, as we did in *Step 4*. We should get a response with the new OU as the parent:

```
{
  "Parents": [
    {
      "Id": "ou-bim7-0s1nqy2w",
      "Type": "ORGANIZATIONAL_UNIT"
    }
  ]
}
```

Figure 1.19 – The response to the `list-parents` subcommand

7. We can list all the OUs under the root OU using the `list-children` subcommand with the child type set to `ORGANIZATIONAL_UNIT`:

```
aws organizations list-children --parent-id r-bim7 --child-type
ORGANIZATIONAL_UNIT --profile awssecadmin
```

This should give us a response similar to the following if we have a total of two OUs, assuming that we created one in the previous recipe:

```
{
  "Children": [
    {
      "Id": "ou-bim7-xmvfviti",
      "Type": "ORGANIZATIONAL_UNIT"
    },
    {
      "Id": "ou-bim7-0s1nqy2w",
      "Type": "ORGANIZATIONAL_UNIT"
    }
  ]
}
```

Figure 1.20 – The response to the `list-children` subcommand

To get the details of the OU, along with its name, we can use the `describe-organizational-unit` subcommand with a single parameter named `organizational-unit-id` by passing in the ID.

## How it works...

From the Management Console, we created an AWS Organization, created OUs under it, and added accounts under the OUs. An OU called `Root` was created by default. The account used to create the Organization is called the management account (formerly known as the **master account**) and is created under the root OU.



We can only initiate the creation of a new organization from an AWS account that is not a member of any organization. We cannot make another AWS account into a management account later, and hence, the account from which we create an organization needs to be selected carefully. We can move an account to any OU, including the root OU. We can also create sub-OUTs within an OU.

AWS Organizations' delegated administrator feature allows specific AWS services, such as AWS IAM Identity Center, to designate a member account within the organization as an administrator for managing that service across all accounts. This enables different teams to manage AWS services using separate accounts tailored to their roles and responsibilities. Services currently supporting this feature include AWS IAM Identity Center, AWS Config, AWS Firewall Manager, Amazon GuardDuty, AWS IAM Access Analyzer, Amazon Macie, AWS Security Hub, Amazon Detective, AWS Audit Manager, Amazon Inspector, and AWS Systems Manager.

From the CLI, we created an AWS account using the `create-account` subcommand. This command returns immediately with a request ID and works asynchronously. We can check the status of our request using the `describe-create-account-status` subcommand by providing the request ID. To check whether an account was created, we can check the **AWS CloudTrail log** for the `CreateAccountResult` event.

The `create-account` subcommand also accepts other parameters, namely `role-name` and `iam-user-access-to-billing`. The `role-name` parameter is used to specify the name of an IAM role that will be automatically pre-configured in the new member account. This role provides administrator permissions to the member account and trusts the management account. This means that users in the management account can assume the role, provided the management account administrator allows this. The default value is `OrganizationAccountAccessRole`. If we log in to the child account and check the `OrganizationAccountAccessRole` role, we will see that it has the `Administrator Access` policy attached to it. If we check the **Trust relationships** section, we will see that our management account has been added as a trusted entity. An administrator from the management account can now switch roles to the child account and have administrator access. For non-admin users to assume the `OrganizationAccountAccessRole` role in the child account and switch roles to log into the child account, the user should be given the `AssumeRole` permission for the role.

The `iam-user-access-to-billing` parameter has to be set to **ALLOW** for IAM users to access account billing information. If it is set to **DENY**, only the root user can access account billing information. The default value is **ALLOW**. We also created an OU and moved our account under the OU. Within the examples, we used the `list-children` subcommand with the `ORGANIZATIONAL_UNIT` child type to list all the OUTs under the root. We can set `child-type` to `ACCOUNT` to list all the accounts instead.

## There's more...

Let's quickly go through some important details about the AWS Organizations service:

- The AWS Organizations service is supported in all regions; however, the endpoints are located in the US East (N. Virginia) for commercial organizations and AWS GovCloud (US-West) for AWS GovCloud (US) organizations.
- The AWS Organizations service is a global service. We don't have to select or specify any region to create organization entities.
- There is no additional cost for using AWS Organizations.
- The number of accounts we can manage within an AWS Organization varies. We can ask AWS support to increase this limit.
- An account can only be part of one organization at a time and within an organization, an account can only be part of one OU at a time.
- We can nest the OUs and accounts up to five levels (including the root).
- We can use **Service Control Policies (SCPs)** to restrict AWS service actions to root accounts, IAM users, and IAM roles in the accounts of our organization.
- SCPs can only deny access; they cannot allow access.
- When both the **permissions boundary** (an IAM feature) and SCP are present, the action is only allowed if the permission boundary, the SCP, and the identity-based policy all allow the action.
- The current list of supported services that can be integrated with AWS Organizations include AWS Account Management, AWS **Application Migration Service (MGN)**, AWS Artifact, AWS Audit Manager, AWS Backup, AWS CloudFormation Stacksets, AWS CloudTrail, Amazon CloudWatch Events, AWS Compute Optimizer, AWS Config, AWS Control Tower, Amazon Detective, Amazon DevOps Guru, AWS Directory Service, AWS Firewall Manager, Amazon GuardDuty, AWS Health, AWS IAM, IAM Access Analyzer, Amazon Inspector, AWS License Manager, Amazon Macie, AWS Marketplace, AWS Network Manager, AWS Resource Access Manager, AWS Security Hub, Amazon S3 Storage Lens, Amazon Security Lake, AWS Service Catalog, Service Quotas, AWS IAM Identity Center (the successor to AWS SSO), AWS Systems Manager, tag policies, AWS Trusted Advisor, AWS Well-Architected Tool, **Amazon VPC IP Address Manager (IPAM)**, and Amazon VPC Reachability Analyzer. We can enable integration from the supported service's dashboard. For an updated list of services, refer to the *See also* section of this recipe.

Let's also go through some of the useful AWS CLI subcommands for AWS Organizations:

- `create-gov-cloud-account` can be used to create accounts in the AWS GovCloud (US) region if we are authorized to do so.
- `invite-account-to-organization` sends an invitation to another account to join our organization.
- `remove-account-from-organization` removes an account from the organization.
- `create-organization` creates an AWS Organization, while `delete-organization` deletes an AWS Organization.
- `leave-organization` removes an account from its parent organization.
- `create-organizational-unit` creates an OU, while `delete-organizational-unit` deletes an OU. To delete an OU, we must remove all accounts and child OUs.
- `update-organizational-unit` renames an OU.
- `describe-account` retrieves information about that account and should be called from the master account. `describe-organization` retrieves information about the organization. `describe-organizational-unit` retrieves information about an OU.
- `list-accounts` lists all the accounts in the organization. `list-accounts-for-parent` lists the child accounts of the given target root or OU. `list-create-account-status` lists the account creation requests that match the given status. `list-roots` lists the roots that are defined in the current organization.
- `tag-resource` and `untag-resource` can be used for managing tags.

### *Different ways to interact with AWS*

We can interact with AWS in a variety of ways, including, the AWS Management Console, AWS CLI, AWS **Software Development Kits (SDKs)**, AWS CloudFormation, external tools such as Terraform by HashiCorp, direct AWS API calls, AWS Tools for PowerShell, AWS **Cloud Development Kit (CDK)**, and the AWS **Serverless Application Model (SAM)**. Each of these methods offers unique advantages depending on the specific task and the level of automation required.

For the scope of the recipes within this book, we will predominantly focus on the AWS Management Console and the CLI. The Management Console is typically utilized for one-time configurations and activities, providing an intuitive and visual means to manage AWS resources. On the other hand, the CLI is particularly suited for repetitive tasks, enabling automation and scriptability. Gaining proficiency in the CLI not only streamlines our AWS operations and lays a solid foundation but also aids in grasping the nuances of other interaction methods, such as AWS SDKs, CloudFormation, Terraform, and more.

---

## See also

- We can learn more about AWS Organizations including best practices and quotas at <https://www.cloudericks.com/blog/getting-started-with-aws-organizations>.
- To learn about all the commands that we can use to work with the AWS CLI, refer to the AWS CLI documentation for organizations at <https://docs.aws.amazon.com/cli/latest/reference/organizations/index.htm>.
- The services that can be integrated with AWS Organizations can be found at [https://docs.aws.amazon.com/organizations/latest/userguide/orgs\\_integrate\\_services\\_list.html](https://docs.aws.amazon.com/organizations/latest/userguide/orgs_integrate_services_list.html).
- We use AWS CLI within this book. However, if you would like to learn more about CloudShell, you can do this at <https://www.cloudericks.com/blog/getting-started-with-aws-cloudshell>.
- We can learn about using AWS Control Tower, instead of using AWS Organizations directly, at <https://www.cloudericks.com/blog/getting-started-with-aws-control-tower>.

## User management and SSO with IAM Identity Center

In this recipe, we will first enable the IAM Identity Center service (previously AWS SSO) and learn to create users and groups within IAM Identity Center. We will then create a permission set and assign a group to an AWS account along with that permission set. Finally, we will see how to log in to the AWS Management Console and AWS CLI from the AWS access portal using SSO.

SSO is a user authentication process that allows a user to access multiple applications or systems with one set of login credentials. This means that after logging in once, the user can access other AWS accounts and apps without needing to log in again for each system. This simplifies the user experience and enhances security by reducing the number of passwords a user must remember and maintain.

### Getting ready

We need an AWS account with AWS organizations enabled. To set up AWS Organizations, we can follow the *Multi-account management with AWS Organizations* recipe from this chapter.

For working with CLI and IAM Identity Center, we need to install and configure AWS CLI V2 as discussed in the *Technical requirements* section in this chapter.

## How to do it...

First, we will enable the IAM Identity Center. Then, we will create a group and a user and add that user to that group. After that, we will create a permission set and assign access for the group to an AWS account making use of the permission set.

### *Enabling the IAM Identity Center and creating users and groups*

To enable IAM Identity Center and create a group and user, we can proceed with the following steps:

1. Log in to AWS Management Console and go to **IAM Identity Center**.

If we have not enabled IAM Identity Center, we should see a screen like the following:

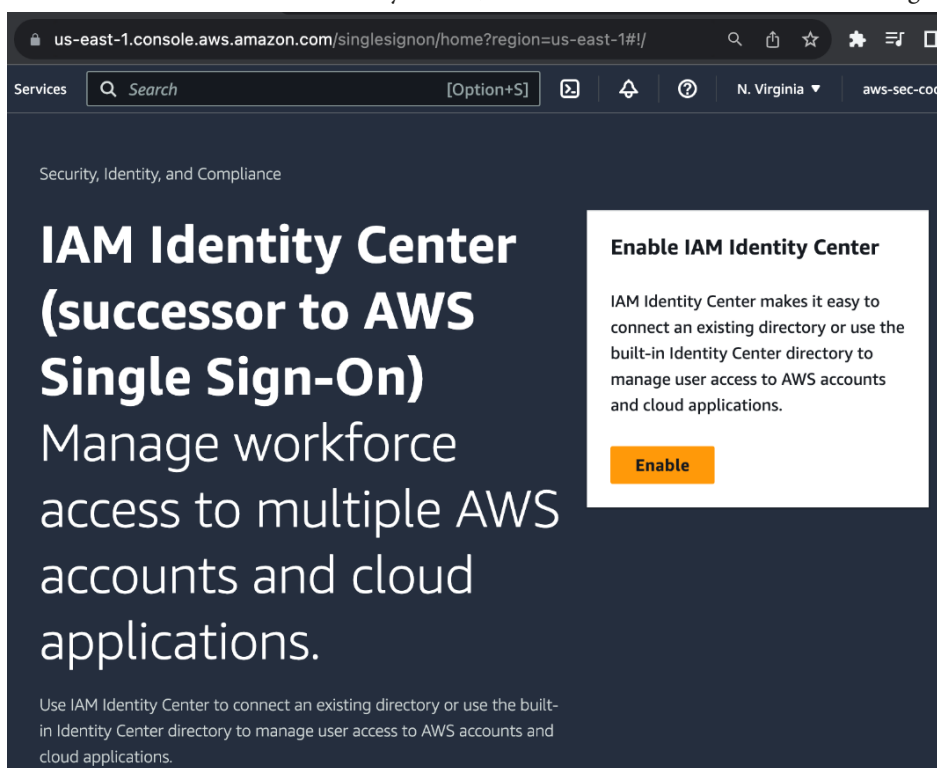


Figure 1.21 – The IAM Identity Center dashboard

2. Click on **Enable** from the IAM Identity Center Dashboard as shown in *Figure 1.21*.

We should now be taken to the **IAM Identity Center** dashboard.

**Dashboard**

IAM Identity Center enables you to manage workforce user access to multiple AWS accounts and cloud applications. [Learn more](#)

### Recommended setup steps

**Step 1**  
[Choose your identity source](#)  
The identity source is where you administer users and groups, and is the service that authenticates your users.

---

**Step 2**  
[Manage access to multiple AWS accounts](#)  
Give users and groups access to specific AWS accounts in your organization.

Or

[Set up Identity Center enabled applications](#)  
Give users and groups access to applications that integrate with your Identity Center directory.

Or

[Manage assignments to your cloud applications](#)  
Give users and groups access to your cloud applications and any SAML 2.0-based custom applications.

### Settings summary

[Go to settings](#)

Identity source  
Identity Center directory

Region  
US East (N. Virginia) | us-east-1

AWS access portal URL  
<https://d-90679fa661.awsapps.com/start>

[Customize](#)

Figure 1.22 – The recommended setup steps for IAM Identity Center

3. Click on **Choose your identity source** under **Step 1** as shown in *Figure 1.22*.
4. Leave the value for **Identity Source** as **Identity Center directory**.
5. For **Attributes for access control**, click **Enable**.

6. Click on **Groups** from the left sidebar and click on **Create group** as shown in the following figure:

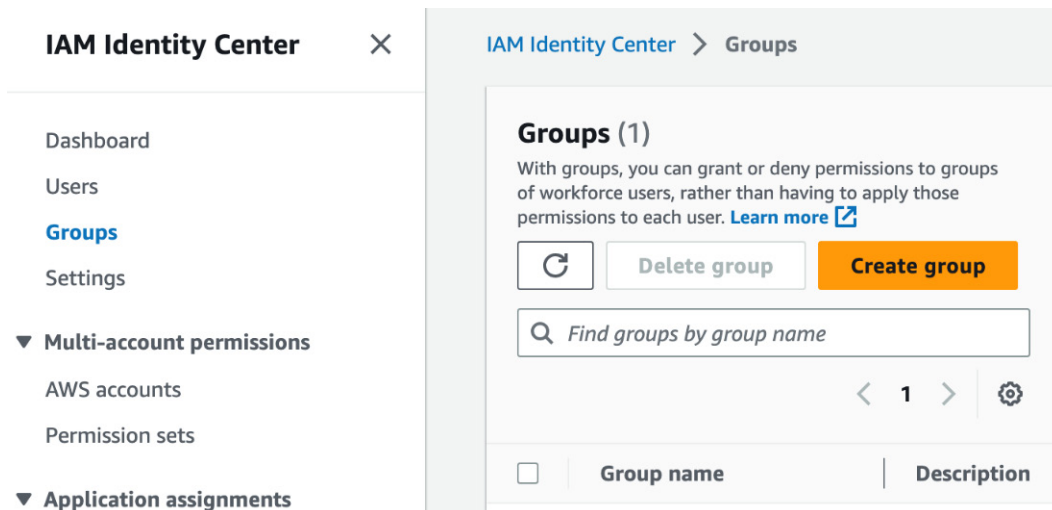


Figure 1.23 – The IAM Identity Center sidebar and Groups page

7. On the **Create group** page, within the **Group details** section, enter **awssecbadmins** as the **Group name** and set **AWS Sec Cookbook Admins group** under **Description**.
8. Scroll down, leave the **Add users to group** section as-is without adding any users for now, and click on **Create group**.
9. Click on **Users** from the left sidebar of IAM Identity Center dashboard and click on **Add user**.
10. In the **Primary information** section, specify **awssecbadmin1** under **Username** and under **Password**, select **Send an email to this user with the password setup instructions**. We can also generate a one-time password and share that with the user instead.

## Specify user details

### Primary information

#### Username

This username will be required for this user to sign in to the AWS access portal. The username can't be changed later.

Maximum length of 128 characters. Can only contain alphanumeric characters or any of the following: +=, @- \_

#### Password

Choose how you want this user to receive their password. [Learn more](#) 

- ☒ Send an email to this user with password setup instructions.
- ☐ Generate a one-time password that you can share with this user.

Figure 1.24 – The Specify user details page in IAM Identity Center

11. Fill in the other fields within the **Primary Information** section, namely **Email address**, **Confirm email address**, **First name**, **Last name**, and **Display name**.
12. Leave the other sections, namely **Contact methods**, **Job-related information**, **Address**, **Preferences**, and **Additional attributes** as-is. Click **Next** on the bottom-right side of the page.
13. On the **Add user to group** page, select the `awsseccbadmins` group that we created earlier in this recipe and click **Next**.
14. On the **Review and add user** page, review the details and click **Add user**. We should now see that the new user is added to the **Users** page.



15. The newly added user needs to check the email and follow the instructions to accept the invitation and complete the password setup.

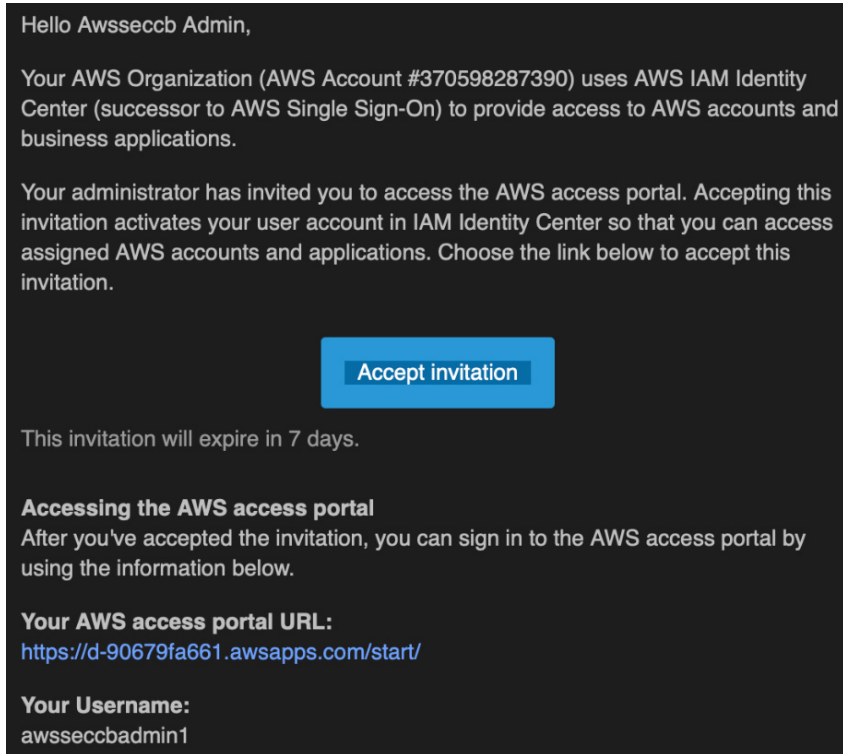


Figure 1.25 – The invitation email for the new user in IAM Identity Center

After completing the instructions, we should be logged in to the AWS access portal, where we can see the applications allocated to us. Currently, since we have not allocated any applications, we should see a **You do not have any applications** message.

Next, we will create a permission set.

### ***Creating a permission set with an AWS managed policy***

Follow the instructions to create a permission set that we can use while assigning access to AWS accounts in the next section:

1. Click on **Permission sets** from the left sidebar as shown in *Figure 1.23*, and on the **Permission sets** page, click on **Create permission set**.
2. Select **Predefined permission set** as the **Permission set type**, and for **Select an AWS managed policy**, select **AdministratorAccess**.

### Permission set type

Types

☒ **Predefined permission set**

Create a predefined permission set by choosing an AWS-defined template. This template enables you to select a single AWS managed policy. For example, you can select a policy that grants permissions for a common job function, such as Billing, or a specific level of access to AWS services and resources, such as ViewOnlyAccess. You can update the permission set as your needs evolve.

☐ **Custom permission set**

Create a custom permission set by selecting AWS managed policies and creating an inline policy (recommended). You can also attach customer managed policies and set a permissions boundary (advanced).

### Policy for predefined permission set

Select an AWS managed policy

☒ **AdministratorAccess**

Provides full access to AWS services and resources.

☐ **Billing**

Figure 1.26 – Permission set type selection in IAM Identity Center

3. Scroll down and click **Next**.
4. On the **Specify permission set details** page, keep the value of **Permission set name** as **AdministratorAccess** and **Session duration** as **1 hour**, add a meaningful description, leave values for other fields empty, and click **Next**.
5. On the **Review and create** page, review everything and click on **Create**.

The new permission set should now appear on the **Permission sets** page.

Next, we will assign the `awssecbadmins` group to an AWS account.

## Providing access to AWS accounts

We can provide access for a group to one or more AWS accounts as follows:

1. Click on the **AWS Accounts** option from the left sidebar as shown in *Figure 1.23* to go to the **AWS accounts** page.
2. Select all the AWS accounts we want to give access to and click on **Assign users or groups**. I have selected the `aws-sec-cookbook-1` account.

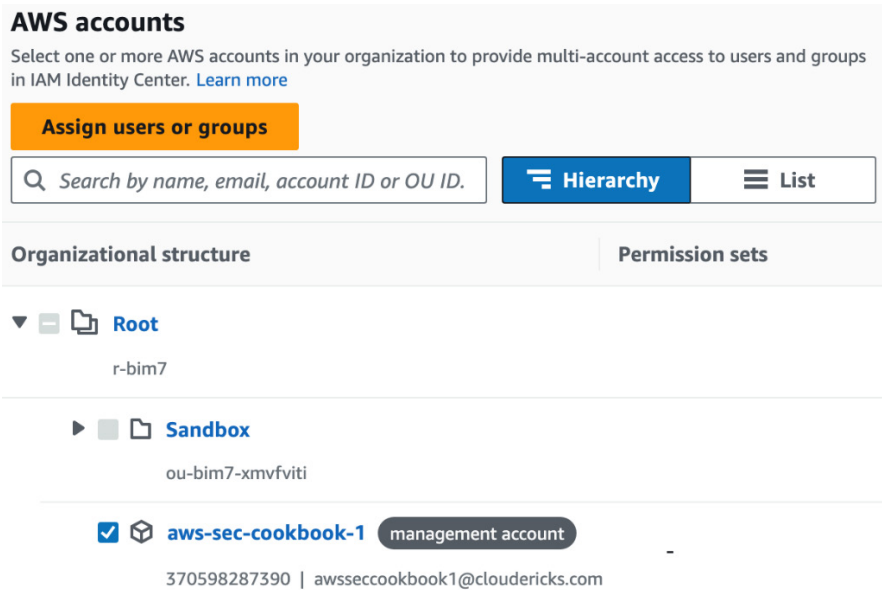


Figure 1.27 – The AWS accounts page in IAM Identity Center

3. On the **Select users and groups** page, in the **Groups** tab, select the `awsseccbadmins` group that we created earlier in this recipe and click **Next** to go to the **Assign permission sets** page.
4. On the **Assign permission sets** page, select the permission set for our group to the selected AWS account. Click **Next** to go to the **Review and submit assignments** page.

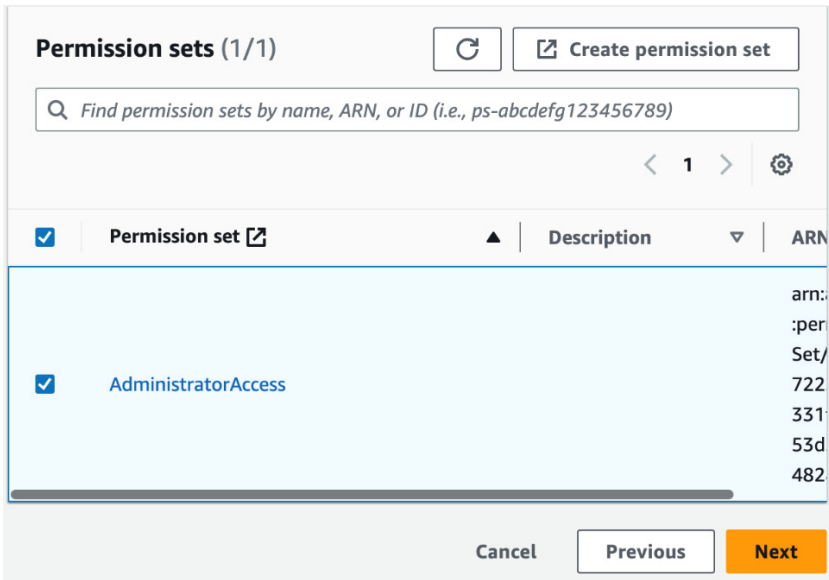


Figure 1.28 – Permission set selection in IAM Identity Center

5. On the **Review and submit assignments** page, review everything and click on **Submit**.
6. Log in to the AWS access portal using the **AWS access portal URL** of AWS Identity Center. We can get the URL from our Identity Center dashboard. It is also present in the invitation email sent to the user's email address when the user was created.

You need to click on **AWS Account (1)** to see the newly assigned AWS account and then click on the account to get the options to log in to **Management Console** or **Command line or programmatic access**, making use of short-term credentials.

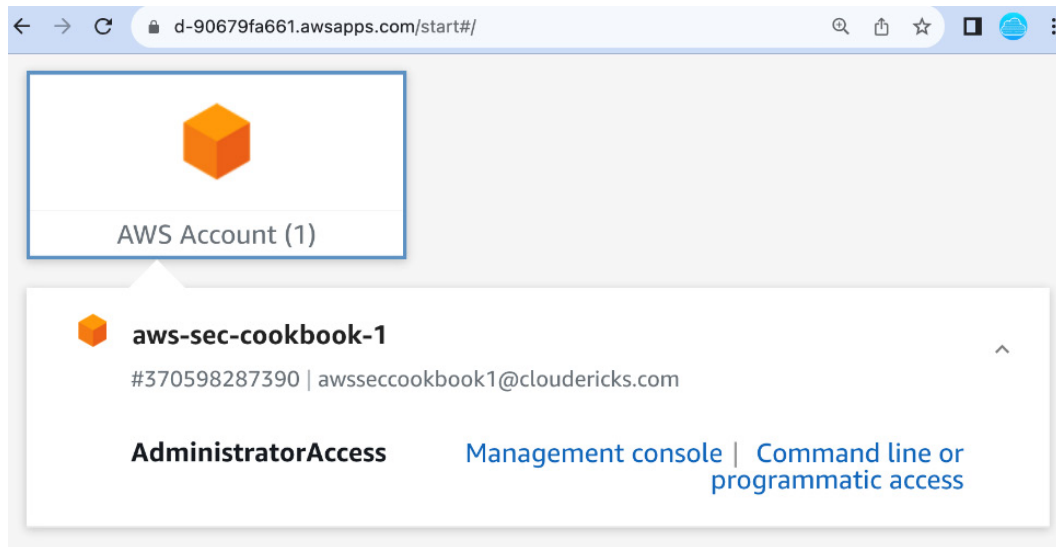


Figure 1.29 – The AWS access portal

7. Click on **Management console** to log in to the AWS account assigned to us. Click on the drop-down menu next to our username to verify the account details.

We can follow the same steps to give access to one or more AWS accounts for users (instead of groups). However, it is a good practice to assign permissions to groups and add or remove users to those groups as needed.

## Configuring SSO for AWS CLI with IAM Identity Center

In the previous section, we saw how to log in to the AWS Management Console using IAM Identity Center from the AWS access portal. In this section, we will learn how to configure AWS CLI V2 with IAM Identity Center:

1. Log in to the AWS access portal as an IAM Identity Center user. We should see a screen like *Figure 1.29*.
2. Click on **Command line or programmatic access** from the AWS access portal instead of the Management Console as we did in *Step 7* of the *Providing access to AWS accounts* sub-section.

We should see a popup that lists different options along with the required steps to work with a CLI.

### Get credentials for AdministratorAccess

---

Create access for the account **aws-sec-cookbook-1 (3705-9828-7390)** with **AdministratorAccess**


Use any of the following options to access AWS resources programmatically or from the AWS CLI. You can retrieve new credentials as often as needed.



[macOS and Linux](#) | [Windows](#) | [PowerShell](#)

---

#### ▼ AWS IAM Identity Center credentials (Recommended)

---

To extend the duration of your credentials, we recommend you configure the AWS CLI to retrieve them automatically using the **aws configure sso**  command. [Learn more](#)

SSO Start URL	<code>https://d-90679fa661.awsapps.com/start#</code>	
SSO Region	<code>us-east-1</code>	

- ▶ Option 1: Set AWS environment variables (Short-term credentials)
- ▶ Option 2: Manually add a profile to your AWS credentials file (Short-term credentials)
- ▶ Option 3: Use individual values in your AWS service client (Short-term credentials)

Figure 1.30 – The steps to work with AWS CLI when using IAM Identity Center

3. Assuming that we have already installed AWS CLI V2, open the command prompt (or terminal), run the `aws configure sso` command, and follow the instructions shown in *Figure 1.30*. Provide **SSO start URL** and **SSO region**, and optionally a name for the SSO session, as shown in the following figure:

```
heartin$aws configure sso
SSO session name (Recommended): cb-demo1
SSO start URL [None]: https://awssecb.awsapps.com/start#
SSO region [None]: us-east-1
SSO registration scopes [sso:account:access]:
Attempting to automatically open the SSO authorization page in your
default browser.
If the browser does not open or you wish to use a different device
to authorize this request, open the following URL:

https://device.sso.us-east-1.amazonaws.com/

Then enter the code:

HDGR-CXDW
The only AWS account available to you is: 370598287390
Using the account ID 370598287390
The only role available to you is: AdministratorAccess
Using the role name "AdministratorAccess"
CLI default client Region [us-east-1]:
CLI default output format [None]: json
CLI profile name [AdministratorAccess-370598287390]: AwsSecCbAdmin

To use this profile, specify the profile name using --profile, as s
hown:

aws s3 ls --profile AwsSecCbAdmin
```

Figure 1.31 – Configuring SSO with AWS CLI V2

Once we have provided values for **SSO session name**, **SSO start URL**, **SSO region**, and **SSO registration scopes**, a browser will open for authorization. Please note that I have provided the customized **SSO start URL** instead of the default one as shown in *Figure 1.30*; both will work. The command prompt will resume once the authorization is complete.

4. Verify by running the `aws s3 ls` command along with the profile name, as shown in *Figure 1.31*.

We only need to configure SSO once for a combination of AWS accounts and roles as explained in the *How it works...* section that follows. Once configured, we can make use of that profile to log in and log out using the `aws sso login` and `aws sso logout` commands respectively, as we will see next.

## Logging in and out for SSO with IAM Identity Center in AWS CLI

We can log in to and out of the AWS account using a configured AWS CLI V2 profile as follows:

1. Log in using the `aws sso login` command, providing the CLI profile name we already configured (as we saw in *Figure 1.31*). Execute the `aws s3 ls` command, providing the same profile name, as follows:

```
heartin$aws sso login --profile AwsSecCbAdmin
Attempting to automatically open the SSO authorization page in your default browser.
If the browser does not open or you wish to use a different device to authorize this request, open the following URL:

https://device.sso.us-east-1.amazonaws.com/

Then enter the code:

HNFV-SHZS
Successfully logged into Start URL: https://awssecb.awsapps.com/start#
heartin$aws s3 ls --profile AwsSecCbAdmin
2023-08-21 19:29:17 awssecbssso
2023-08-01 17:14:40 cloudericks
```

Figure 1.32 – Logging into an AWS account from CLI using SSO

The `aws sso login` command will open a browser for authorization similar to the `aws configure sso` command. The command prompt will resume once the authorization is complete.

### Important note

If we have multiple profiles or if we are not using the default profile, we will need to specify the profile when executing AWS CLI commands even after the SSO login. Also, if we try to use a profile for which we don't have access, we will get an error that says that an error occurred (`ForbiddenException`) when calling the `GetRoleCredentials` operation: No access.

2. Log out using the `aws sso logout` command and execute the `aws s3 ls` command, providing the same profile name as before.

```
heartin$aws sso logout
heartin$aws s3 ls --profile AwsSecCbAdmin

Error loading SSO Token: Token for cb-demo1 does not exist
heartin$
```

Figure 1.33 – Logging out from the AWS account from CLI using SSO

## How it works...

AWS IAM Identity Center is the successor to AWS SSO and helps us securely manage access centrally across multiple AWS accounts and applications. The IAM Identity Center is the suggested method for handling authentication and authorization on AWS. It is suitable for organizations big and small.

From the AWS IAM Identity Center dashboard, we can configure access to AWS accounts within an AWS Organization, numerous cloud applications such as Microsoft 365, salesforce, and so on, EC2 Windows instances, and even other **SAML 2.0-enabled applications**. Once configured, we only need to log in to the AWS access portal and then we can log in to all the configured AWS accounts and applications through SSO without providing any other additional credentials.

An **Identity Provider (IdP)** is a service that stores and verifies user identity. We can make use of an IdP to log in to multiple applications without providing additional credentials using SSO. We used the built-in IdP of the IAM Identity Center within our recipe. However, instead of the built-in IdP, we can also use one of the many supported IdPs such as Microsoft Entra ID (previously Active Directory or Azure AD), Okta, Ping Identity, Jump Cloud, and Google Workspace.

Within this recipe, while enabling IAM Identity Center, we also enabled **attributes for access control**. We can assign users to workloads in AWS based on existing attributes in the user's identity source to control access to resources, and thus implement **Attribute-Based Access Control (ABAC)**. ABAC is a method of regulating access based on attributes (characteristics or properties) associated with users, resources, or the environment. Unlike **Role-Based Access Control (RBAC)**, which grants access based on the roles of users within an organization, ABAC uses a wide range of attributes, such as user location, time of access, and even sensitivity of the accessed resource. This allows for more flexible, context-aware, and policy-driven access control, enabling more granular and dynamic permission management.

We can assign varying permission levels to users or groups for different AWS accounts and applications using permission sets. For example, we can give developers complete access to developer accounts and read-only access to production accounts. We can select the predefined permission set option as we did in this recipe selecting one of the available AWS-managed policies or we can create a custom permission set.



The following screenshot from the IAM Identity Center lists down policies available currently when we select the predefined permission set option.

### Policy for predefined permission set

#### Select an AWS managed policy

- ☒ **AdministratorAccess**  
Provides full access to AWS services and resources.
- ☐ **Billing**  
Grants permissions for billing and cost management. This includes viewing account usage and viewing and modifying budgets and payment methods.
- ☐ **DatabaseAdministrator**  
Grants full access permissions to AWS services and actions required to set up and configure AWS database services.
- ☐ **DataScientist**  
Grants permissions to AWS data analytics services.
- ☐ **NetworkAdministrator**  
Grants full access permissions to AWS services and actions required to set up and configure AWS network resources.
- ☐ **PowerUserAccess**  
Provides full access to AWS services and resources, but does not allow management of Users and groups.
- ☐ **ReadOnlyAccess**  
Provides read-only access to AWS services and resources.
- ☐ **SecurityAudit**  
The security audit template grants access to read security configuration metadata. It is useful for software that audits the configuration of an AWS account.
- ☐ **SupportUser**  
This policy grants permissions to troubleshoot and resolve issues in an AWS account. This policy also enables the user to contact AWS support to create and manage cases.
- ☐ **SystemAdministrator**  
Grants full access permissions necessary for resources required for application and development operations.
- ☐ **ViewOnlyAccess**  
This policy grants permissions to view resources and basic metadata across all AWS services.

Figure 1.34 – Policies for the predefined permission set

With the custom permission set option, we can choose from an AWS-managed policy, a customer-managed policy, and an inline policy, and even optionally set a permissions boundary. We will see a custom permission set in the *Creating customer-managed policies in IAM Identity Center* recipe in *Chapter 2*.

In the recipe, our user had one permission set assigned to one AWS account. If we have users with multiple permission sets and access to multiple AWS accounts, we can choose the AWS account and the permission set to log in from the AWS access portal, as shown in the following figure:

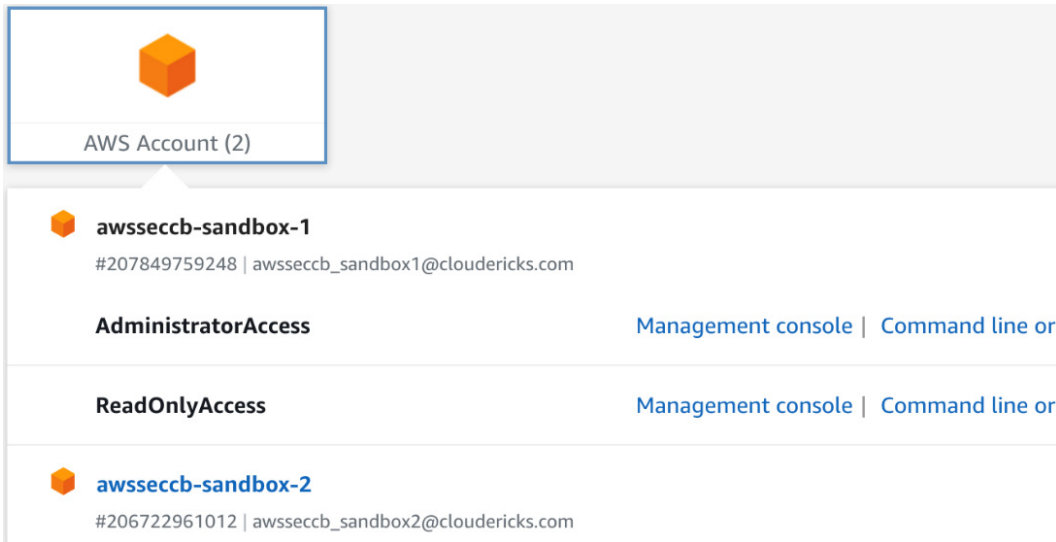


Figure 1.35 – The AWS access portal with multiple AWS accounts and permissions

Even when configuring AWS SSO for CLI, we will be given options to select the AWS account and permission set. First, we will be asked to select the AWS account as follows:

```
There are 2 AWS accounts available to you.
awsseccb-sandbox-2, awsseccb_sandbox2@cloudericks.com (2067...
> awsseccb-sandbox-1, awsseccb_sandbox1@cloudericks.com (2078...
```

Figure 1.36 – AWS CLI account selection

After we select the account, we will be given the option to choose the role (which is based on the permission set) if we have multiple roles available to choose from in the selected AWS account.

```
Using the account ID 207849759248
There are 2 roles available to you.
> ReadOnlyAccess
AdministratorAccess
```

Figure 1.37 – AWS CLI role selection

## There's more...

We can administer IAM Identity Center from only the management account of the Organization or a member account registered as an IAM Identity Center delegated administrator. Otherwise, we will see an error message that looks like the following:

### **You can't administer IAM Identity Center from this member account**

IAM Identity Center is currently configured in the US East (N. Virginia) Region. To administer IAM Identity Center, do either of the following:

- Sign in to the management account for your organization (370598287390).
- Sign in to a member account that is registered as an IAM Identity Center delegated administrator.

For more information, see [\[link\]](#).

Figure 1.38 – An error message for a member account that is not a delegated administrator

We can make a member account as a delegated administrator as follows:

1. Log in to the AWS Management Console of the management account of our organization and go to the **IAM Identity Center** dashboard.
2. Click on **Settings** from the left sidebar of IAM Identity Center.
3. Go to the **Management** tab.
4. Find the **Delegated administrator** section and click on **Register account**. This will show us the organizational structure of our AWS Organization.
5. Under the **Organizational structure** section, select the member account we want to make a delegated administrator and click on **Register account**.

We should see a message that the member account was registered successfully as an IAM Identity Center delegated administrator. It might take some time to grant administrative access to the member account.

We can customize the default AWS access portal URL (e.g., `https://d-90679fa661.awsapps.com/start/`) to one that uses a custom subdomain (e.g., `https://awsseccb.awsapps.com/start`) to make it more memorable for our users. For customizing the URL, we can click on the **Customize** button from the **Settings summary** section on the right side of the IAM Identity Center dashboard, as shown in *Figure 1.22*, and configure a custom subdomain. Once it has been customized, we won't be able to change it again.

---

## See also

- We can explore more about IAM Identity Center at <https://www.cloudericks.com/blog/demystifying-aws-iam-identity-center-formerly-aws-sso>.
- We can learn more about AWS CLI at <https://www.cloudericks.com/blog/getting-started-with-aws-cli>.



# Access Management with IAM

## Policies and Roles

Secure access management is crucial for effectively managing who can access our AWS resources and what actions they can perform. This knowledge ensures that our AWS environment remains secure and compliant by allowing us to precisely control permissions and minimize the risk of unauthorized access. It empowers us to implement a robust security posture that safeguards our data and resources, while also enabling a scalable and efficient way to manage access for users and services across our AWS infrastructure. In this chapter, we will learn about secure access management within the AWS cloud using **IAM policies** and **IAM roles**.

AWS supports various policy types, such as **identity-based policies**, **resource-based policies**, **session policies**, **permissions boundaries**, **service control policies (SCPs)**, and **access control lists (ACLs)**. While we will learn most of these policy types with detailed recipes, ACLs are only discussed theoretically since ACLs are not recommended to be used anymore. We will then learn about IAM roles and how to use them for implementing the cross-account access needed to implement identity account architecture, and how to use roles for cross-service access to allow one AWS service to access another service securely. We will continue to explore IAM policies and roles in future chapters within this book.

This chapter will cover the following recipes:

- Creating a customer-managed IAM policy
- Using policy variables within IAM policies
- Creating customer-managed policies in IAM Identity Center
- Setting IAM permission boundaries for IAM entities
- Centralizing governance in AWS Organizations with SCPs
- IAM cross-account role switching and identity account architecture
- Cross-service access via IAM roles on EC2 instances

## Technical requirements

Before diving into the recipes of this chapter, we need to ensure we have the following requirements in place:

- We need an active AWS account to complete most of the recipes within this chapter. We can use an account that is part of an AWS organization or a standalone account for most of the recipes within this chapter. I will be using the `awssecb-sandbox-1` AWS organization **member account** that we created in the *Multi-account management with AWS Organizations* recipe from *Chapter 1*. However, unless specified otherwise, I won't be utilizing AWS Organizations features, meaning you can follow these steps with a standalone account, too. Note that certain recipes may have different AWS account requirements, which will be specified in those recipes.
- For administrative actions, we need a user who has `AdministratorAccess` permission to the AWS account we are working with. This can be an IAM Identity Center user or an IAM user. I will be using the `awssecbadmin1` IAM Identity Center user we created in the *User management and SSO with IAM Identity Center* recipe from *Chapter 1*. However, unless specified otherwise, I won't be utilizing any IAM Identity Center features, meaning you can follow these steps with an IAM user, too, if the user has the required permissions in the account we are working with. If you are using an IAM user, you can create the user following the *Setting up IAM, account aliases, and billing alerts* recipe from *Chapter 1*. Note that certain recipes may have specific user requirements, which will be specified in those recipes.
- To execute AWS **Command Line Interface (CLI)** commands on a local machine or a virtual machine, we need AWS CLI v2 installed. Configure the AWS CLI using IAM access keys for IAM users or utilize temporary credentials from the IAM Identity Center for IAM Identity Center users. *Chapter 1* details how to set up the CLI using both IAM and IAM Identity Center. Unless specified, we can manage users with either IAM or IAM Identity Center, though AWS recommends using IAM Identity Center. Additionally, **AWS CloudShell**, available through the AWS Console, offers a convenient way to perform most CLI commands discussed in this chapter. With both IAM and IAM Identity Center, we need to create a profile per user per account. Therefore, it is good to include the account name as part of the profile name. For instance, for the administrator user for the `awssecb-sandbox-1` sandbox account, we can set up a `CLI Sandbox1Admin1` profile: `awssecbadmin1`. For IAM Identity Center, we need to create a profile per user per account per permission set assignment, and hence we could include details of the assignment also in its name.
- For certain recipes, we will need an Amazon **Simple Storage Service (S3)** bucket to test the policies. Hence, a basic understanding of Amazon S3 service is recommended. Unless otherwise specified, we can create S3 buckets with the following configuration: For **Bucket name**, give a unique name as every bucket needs to have a unique name across AWS accounts. For **Object Ownership**, select **ACLs disabled (recommended)**. For **Block Public Access settings for this bucket**, select **Block all public access**. For **Bucket Versioning**, select **Disable**. For **Default encryption**, select **Server-side encryption with Amazon S3 managed keys (SSE-S3)**. For **Object Lock under Advanced settings**, select **Disable**.

The code files for this book are available at <https://github.com/PacktPublishing/AWS-Security-Cookbook-Second-Edition>. The code files for this chapter are available at <https://github.com/PacktPublishing/AWS-Security-Cookbook-Second-Edition/tree/main/Chapter02>.

## Creating a customer-managed IAM policy

In this recipe, we will create a customer-managed identity-based IAM policy to manage access to an S3 bucket. We will give permission to list all S3 buckets and will further use the `Condition` policy element to restrict the permission based on the requester's IP address. We will use the AWS Management Console for this recipe, but you may do it from the AWS CLI by making use of the provided JSON code following the next recipe titled *Using policy variables within IAM policies* from this chapter.

IAM policies can work with both IAM and IAM Identity Center. Within this recipe, we will use the IAM policies with IAM entities. In the recipe titled *Creating customer-managed policies in IAM Identity Center* from this chapter, we will learn how to use the same policy with IAM Identity Center entities.

### Getting ready

We need the following to successfully complete this recipe:

- A working AWS account, `awsseccb-sandbox-1`, and a user with `AdministratorAccess` permission to that account, `awsseccbadm1n1`, following the *Technical requirements* section of this chapter.
- For testing this recipe, we need an IAM user named `awsseccb_iam_user1` with minimal or no permission and an S3 bucket, following the *Technical requirements* section of this chapter.

### How to do it...

We will first create an IAM policy using the `awsseccbadm1n1` administrator user and then attach it to the `awsseccb_iam_user1` IAM user.

#### *Creating a customer-managed IAM policy from the AWS Management Console*

We can create a policy using the IAM from the AWS Management Console as follows:

1. Log in to the console of `awsseccb-sandbox-1` as the `awsseccbadm1n1` user with `AdministratorAccess` permission and go to the **IAM** dashboard.
2. Click on **Policies** from the left sidebar.



3. Click on **Create Policy**. This will provide us with a visual editor as shown in the following figure.

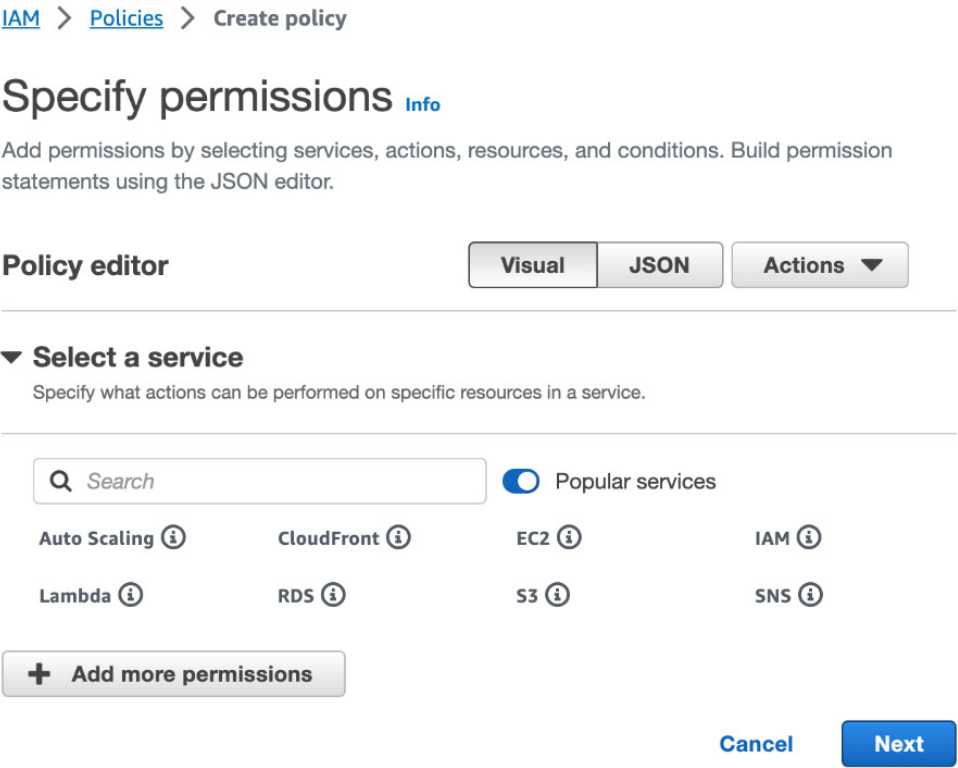


Figure 2.1 – Specifying permissions for policy

If we have already created the policy in JSON format, we can click on the **JSON** tab, enter it directly, and save it. In that case, we can skip *Steps 4* to 6.

4. Under the **Select a service** section, click on **S3**. We should now see a screen like the following:

▼ S3

Allow

0 Actions

+

+

+

+

Specify what actions can be performed on specific resources in S3.

▼ Actions allowed

Specify actions from the service to be allowed.

Q

Filter Actions

[Switch to deny permissions](#)

i

Manual actions | [Add actions](#)

☐ All S3 actions (s3:\*)

Access level

[Expand all](#) | [Collapse all](#)

► List (10)

► Read (53)

► Write (42)

► Permissions management (15)

► Tagging (10)

► Resources

Specify resource ARNs for these actions.

► Request conditions - *optional*

Actions on resources are allowed or denied only when these conditions are met.

+ Add more permissions

Cancel

Next

Figure 2.2 – Configuring permissions for S3 resource

- Under the **Access level** heading, expand **List** and select `ListAllMyBuckets` as shown in the following figure:

Access level

[Expand all](#) | [Collapse all](#)

▼ List (Selected 1/10)

☐ All list actions

☐ ListAccessPoints 

i

☐ ListAccessPointsForOb 

i

☒ ListAllMyBuckets 

i

☐ ListBucket 

i

☐ ListBucketMultipartUp 

i

☐ ListBucketVersions 

i

☐ ListJobs 

i

☐ ListMultipartUploadPa 

i

☐ ListMultiRegionAccess 

i

☐ ListStorageLensConfig 

i

urations

Figure 2.3 – Selecting access level permission

**Important note**

In the Resources section, the wildcard character (\*) will be chosen to signify all resources since the `ListAllMyBuckets` action isn't specific to any particular resource. If an action such as `ListBucket`, which is specific to a particular bucket, is chosen and we desire to limit it to a single bucket, we need to select **Specific**, click on **Add ARN**, and input the **Amazon Resource Name (ARN)** of our targeted bucket, following the format `arn:aws:s3:::<bucket_name>`. An ARN is a unique identifier that AWS uses to specify resources within its cloud services.


6. Under the **Request conditions** section, select **Requested from IP**, provide our IP address, and click **Add Ip**. We can also specify the IP address range in the **Classless Inter-Domain Routing (CIDR)** format as shown in the following figure.

▼ **Request conditions - optional**


Actions on resources are allowed or denied only when these conditions are met.

☐ **User is MFA Authenticated**

Filters access if MFA was used to validate the temporary security credentials that made the request



☒ **Requested from IP**





Filters access by the requester's IP address 

Example: 210.75.12.75/16

**Add Ip**

**+ Add another condition**

**+ Add more permissions**

 Security: 0  Errors: 0  Warnings: 0  **Suggestions: 2**

**Cancel** **Next**

Figure 2.4 – Configuring request conditions

- Optionally, we can click **+ Add another condition** to add additional conditions as needed.
7. Click **Next** to go to the **Review and create** page.
8. Enter `MyS3ListAllBucketsPolicy` in the **Policy name** field and `My S3 List All Buckets Policy` in the **Description – optional** field (or any name and description we want).

9. Scroll down to review the policy details and click **Create policy**.

10. Go to the policy and verify the policy JSON that was generated from the **JSON** tab:

**Permissions defined in this policy** [Info](#)

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it.

[Copy](#) [Edit](#)

[Summary](#) [JSON](#)

```

1  {
2    "Version": "2012-10-17",
3    "Statement": [
4      {
5        "Sid": "VisualEditor0",
6        "Effect": "Allow",
7        "Action": "s3:ListAllMyBuckets",
8        "Resource": "*",
9        "Condition": {
10       "IpAddress": {
11         "aws:SourceIp": "210.75.12.75/16"
12       }
13     }
14   }
15 ]
16 }
```

Figure 2.5 – Generated policy as JSON

Please make sure the IP address is yours and not copied from the sample code.

We can follow the *Attaching IAM policies to IAM group via AWS Management Console* subsection to assign the policy we created to an IAM group. If we are using IAM Identity Center, we can associate the policy with an IAM Identity Center group following the *Creating Customer-Managed Policies in IAM Identity Center* recipe from this chapter.

### Attaching IAM policies to IAM group via AWS Management Console

It is good practice to add permissions to a user through a group. Hence, create a group called `awssecbusers` and add our `awssecb_iam_user1` user to that group. We can assign permissions to the IAM group as follows:

1. Log in to the console of `awssecb-sandbox-1` as the `awssecbadmin1` administrator user and go to the **IAM** dashboard.
2. Click on **User groups** from the left sidebar of the **IAM** dashboard.
3. Click on the `awssecbusers` group to go to the group's page.

4. Go to the **Permissions** tab, and from the **Add permissions** dropdown menu, click **Attach policies**.

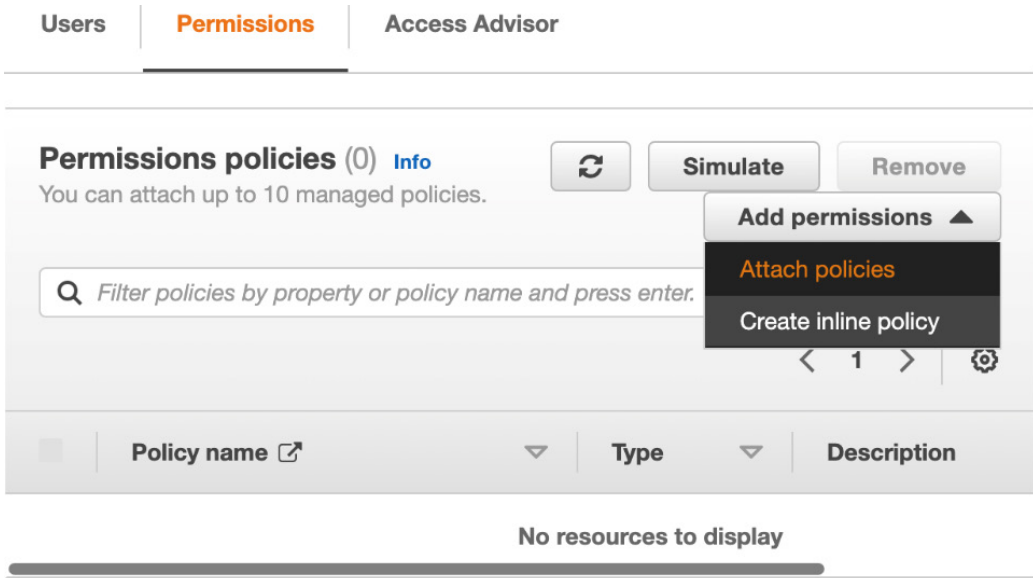


Figure 2.6 – Configuring permissions for group

5. In the **Add permissions** page, select the `MyS3ListAllBucketsPolicy` policy, scroll down, and click on **Add permissions**.

We should now be able to see the new policy added to the group under the **Permissions** tab.

**Important note**

We can also attach policies to groups or users from the **Policy** tab of the **IAM** dashboard.

6. We can now verify the changes as follows:
- I. Log into the AWS Management Console as the `awsseccb_iam_user1` user using the sign-in URL for IAM users for our account as we learned in the *Setting up IAM, account aliases, and billing alerts* recipe in *Chapter 1*.
  - II. Go to the **S3** service.
  - III. Click on **Buckets** from the left sidebar.

We should be able to see all the buckets:

Buckets (1) Info

Buckets are containers for data stored in S3. [Learn more](#)

↺

Copy ARN

Empty

Delete

Create bucket

🔍 Find buckets by name

< 1 >

⚙️

	Name ▲	AWS Region ▼	Access ▼	Creation date ▼
<input type="radio"/>	awsseccb	US East (N. Virginia) us-east-1	⊗ Insufficient permissions	August 13, 2023, 16:38:07 (UTC+05:30)

Figure 2.7 – Listing buckets in Amazon S3

We had given only the `ListAllMyBuckets` permission and hence we will see **Insufficient permissions** under the **Access** column.

In this section, we learned how to assign an IAM policy to an IAM group. However, it is recommended to use IAM policies with IAM Identity Center, which we will explore in the upcoming recipe.

## How it works...

In this recipe, we created a customer-managed IAM policy. There are different IAM policy types, as we saw in the chapter introduction. We explored an identity-based policy within this recipe. Identity-based policies define permissions for IAM entities such as users, groups, or roles. Identity-based policies can be either managed policies or inline policies. Managed policies are policies that can be reused by associating them with multiple IAM entities and inline policies are attached directly to an IAM entity.

Managed policies can be further divided into AWS-managed and customer-managed policies. AWS-managed policies are created and maintained by AWS. While we can view and use them, we cannot edit them. Customer-managed policies are policies that we create for our needs when the AWS-managed policies are not sufficient. Even though we can directly assign the managed policies to a user, it is recommended always to assign them to a group and then add users to that group. Inline policies are attached directly to an IAM entity and cannot be reused.

## Understanding the IAM policy structure

**IAM policies** are JSON documents and follow a structure that is followed by most policy types within AWS, except ACLs, which are XML-based, and **virtual private cloud (VPC)** endpoints. The following is an example of an IAM policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": "s3:ListBucket",
      "Resource": "arn:aws:s3:::awssecb",
      "Condition": {
        "IpAddress": {
          "aws:SourceIp": "210.75.12.75/16"
        }
      }
    }
  ]
}
```

Figure 2.8 – An IAM policy example

Here are the main elements of the IAM policy document:

- The **Version** element Indicates the policy language version. In *Figure 2.8*, we are using the *2012-10-17* version of the policy language, which is the most recent version.
- The optional **Id** element specifies an identifier for the policy, primarily used to distinguish between different versions or instances of a policy. Here is an example: "**Id**" : "S3ReadOnlyPolicyID1".
- The **Statement** element contains the main policy information, including permissions, resources, and conditions. Statements are added as arrays to the **Statement** element.

A **Statement** element for an IAM policy may contain the following sub-elements:

- The optional **Sid** element represents the statement ID. This can be used to provide a description of the policy.
- The **Effect** element specifies whether we want to allow or deny access to a resource. The supported values are **Allow** and **Deny**.
- The **Action** element specifies the permission or permissions (`s3:ListAllMyBuckets`) for which this statement should be applied. We can also specify **\*** to denote any action.

- The optional `NotAction` element is the opposite of the `Action` element and specifies that the policy allows or denies all actions except those listed. The `Action` and `NotAction` elements can be included in the same overall IAM policy but not within the same individual statement of that policy. Each statement in the policy should either specify `Action` or `NotAction`.
- The `Resource` element specifies the ARN of the resource (for example, S3 bucket) that the statement is applied to. For S3 buckets, ARN should follow the following format: `arn:aws:s3:::<bucket_name>/<key_name>`. To specify multiple values, use a comma to separate them. We can specify `*` to denote any resources. In this recipe, we used the `ListAllMyBuckets` policy with `Resource` as `*` as the `ListAllMyBuckets` policy is not a resource-specific policy. We want to use a resource-specific policy such as `ListBucket` and restrict it to a bucket, as shown in *Figure 2.8*.
- The optional `NotResource` element is the opposite of `Resource` and specifies all resources except the listed resources to which the actions apply. A single statement will either contain `Resource` or `NotResource` but not both. However, they can be included in the same overall IAM policy as part of two different statements.
- The `Principal` element identifies AWS users, accounts, services, or other entities that are granted or denied access to a resource. It is predominantly used in resource-based policies to specify which entities are allowed or denied access to a specific AWS resource. Identity-based policies do not use the `Principal` element, as these policies are directly attached to users, groups, or roles, thus rendering the inclusion of a `Principal` element unnecessary. Permissions boundaries also do not include the `Principal` element; instead, they serve as a ceiling to the permissions that IAM entities (including users and roles but excluding groups) can have. SCPs also do not include the `Principal` element. Following is an example for a `Principal` element: `"Principal": { "AWS": "arn:aws:iam::Account-ID-without-hyphens:user/Rick" }`.
- The optional `NotPrincipal` element is the opposite of the `Principal` element and specifies all principals except the listed principals who are allowed or denied by the policy. Similar to the `Principal` element, we cannot have it within SCPs or permissions boundaries. We can use one or the other of `Principal` and `NotPrincipal` within a single policy statement but not both. They can, however, be used in separate statements within the same overall policy.
- The optional `Condition` element allows us to conditionally execute policies as we can see from *Figure 2.8*. We use Boolean operators to match the condition against values in the request. For example, in this recipe, we used the `IpAddress` condition along with the `aws:SourceIp` parameter to allow the actions only if a request is made from the specified IP address. We can also specify a range of IP addresses using the CIDR notation. CIDR is a method for allocating and routing IP addresses without relying on traditional fixed class-based systems. The following are some of the predefined condition keys that are supported by all AWS services that support IAM access control: `aws:CurrentTime`, `aws:EpochTime`, `aws:MultiFactorAuthAge`, `aws:MultiFactorAuthPresent`, `aws:PrincipalOrgID`, `aws:PrincipalArn`, `aws:RequestedRegion`, `aws:SecureTransport`, and `aws:UserAgent`.



## There's more...

Let's go through some core concepts related to policies in AWS, including AWS policy types and the policy evaluation logic.

### *AWS policy types*

The following is a list of important policy types in AWS along with examples and use cases:

- **Identity-based policies:**
  - **Example:** IAM policies attached to users, groups, or roles.
  - **Use case:** They allow a developer read-only access to a specific S3 bucket.
- **Resource-based policies:**
  - **Example:** S3 bucket policies and IAM role trust policies.
  - **Use case:** They grant a user in another AWS account read and write access to a specified bucket.
- **Session policies:**
  - **Example:** A policy limiting permissions when assuming a role temporarily.
  - **Use case:** They provide a developer with temporary enhanced permissions for troubleshooting while limiting access to read-only to ensure security.
- **Permissions boundaries:**
  - **Example:** They set a maximum permission limit for IAM users or roles.
  - **Use case:** They prevent an IAM user from escalating their own permissions beyond a defined boundary to ensure security and compliance.

The following elements are not supported in permissions boundaries: `Principal`, `NotPrincipal`, and `NotResource`.

Permissions boundaries operate as a mechanism for limiting the maximum permissions that can be granted by any attached policies, ensuring a layer of security and control over permissions allocation. The permissions boundary does not grant access. In essence, if an identity-based policy allows certain actions, those actions cannot be performed unless they are also within the confines of the permissions boundary. If an identity-based policy attempts to grant permissions beyond what is allowed by the permissions boundary, those excess permissions will be restricted. In other words, effective permissions are the intersection of the identity-based policy and the permissions boundary.

- **SCPs:**

- **Example:** A policy that restricts the deletion of IAM roles across all accounts in an AWS organization.
- **Use case:** They enhance security by ensuring that certain IAM entities and policies are not modified or deleted.

The following elements are not supported in SCPs: `Principal`, `NotPrincipal`, and `NotResource`.

SCPs in AWS Organizations function similarly to permission boundaries, serving as guardrails that restrict the maximum permissions for IAM entities within AWS accounts in the organization. SCPs don't grant permissions but limit them, working alongside IAM permission policies. Effective permissions are an intersection of the allowances of IAM policies and SCPs, and any explicit `Deny` in either overrides the allowed permissions. Thus, IAM entities can only perform actions allowed by both IAM policies and SCPs.

- **ACLs:**

- **Example:** They specify which principals in another AWS account can access a specific S3 bucket.
- **Use case:** They enable secure data sharing between AWS accounts by granting specific permissions to access data stored in an S3 bucket.

Each policy type serves as an essential tool in the AWS environment, contributing to the intricate ecosystem of access control and security, ensuring optimal operational efficiency and security. We discussed identity-based policies in this chapter and will be discussing more variations of them and other policy types within subsequent recipes within this book. Next, we will look into an overview of the IAM policy evaluation logic.

### ***IAM policy evaluation logic***

**AWS IAM policy evaluation logic** determines whether a request is allowed or denied based on the policies attached to the IAM principal (user or role) making the request, and any permissions boundaries or resource-based policies that apply.

When SCPs and permissions boundaries are not present, a union of all permissions is considered for evaluation. Here's a brief outline of the steps involved in the evaluation process:

- The evaluation logic checks for any explicit `Deny` statements that apply to the request. If there's at least one explicit `Deny`, the request is denied irrespective of any `Allow` statements.
- If there are no explicit `Deny` statements, the logic checks for any explicit `Allow` statements. If there's at least one `Allow`, the request is allowed, provided there isn't an explicit `Deny`.
- If there are no explicit `Allow` or `Deny` statements that apply to the request, the request is denied by default because IAM is a default `Deny`, explicit `Allow` system.

If we use SCPs or permissions boundaries, an intersection of other policies with SCPs or permissions boundaries is considered for evaluation. This is because SCPs and permissions boundaries set the maximum permissions you can have. Therefore, if an SCP or permission boundary gives permission for S3 access and an identity policy gives permission for EC2, our effective permission will be none. To understand SCPs better, we can refer to the discussion about SCPs within the *Multi-account management with AWS Organizations* recipe in *Chapter 1*.

Next, we will look into some additional concepts related to IAM policies.

### ***Additional notes on IAM policies***

Let's quickly go through some additional concepts related to policies in AWS:

- **AWS managed - job function** type is a subset of **AWS managed** type and is designed to align with common IT job functions. The current list of job functions includes administrator, billing, database administrator, data scientist, developer power user, network administrator, read-only access, security auditor, support user, system administrator, and view-only user.
- The **AWS Policy Generator** can generate the following policy types: **Simple Queue Service (SQS)** queue policy, S3 bucket policy, VPC endpoint policy, IAM policy, and **Simple Notification Service (SNS)** topic policy. It is currently available at <https://awspolicygen.s3.amazonaws.com/policygen.html>.
- If both **Allow** and **Deny** effects are set for the same action and resource in a policy, **Deny** will always override **Allow**.
- An IAM policy cannot be used to grant permissions for anonymous users, unlike **S3 ACL** and the **Bucket policy**.
- The IAM policy cannot be applied to a root user and can only be applied to IAM users.

To truly master and make the most out of AWS IAM, it is essential to grasp the finer details of AWS policies. With a comprehensive understanding of policy management and policy evaluation logic, we can try to strike a balance between security and operational efficiency. For the latest on policy types, evaluation rules, and best practices, always refer to the official AWS documentation.

### **See also**

We can continue learning about AWS policies and permissions here: <https://www.cloudericks.com/blog/demystifying-aws-policies-and-permissions>.

## **Using policy variables within IAM policies**

**IAM policy variables** are a set of predefined placeholders that we can use in IAM policy documents, and they get replaced with the actual values at runtime. They help in creating more dynamic and flexible policies. In this recipe, we will create an S3 bucket with folders matching the usernames of

IAM users. With the help of the `${aws:username}` policy variable, we will allow the IAM user to list the contents of only the folder with their names. We will use the AWS CLI for this recipe, but you may do it from the AWS Management Console by making use of the provided JSON code following the previous recipe.

## Getting ready

We need the following to successfully complete this recipe:

- A working `awssecbb-sandbox-1` AWS account, a user with `AdministratorAccess` permission to that account, `awssecbbadmin1`, and a corresponding `Sandbox1Admin1CLI` profile, following the *Technical requirements* section of this chapter.
- For testing this recipe, we need two IAM users, `awssecbb_iam_user1` and `awssecbb_iam_user2`, with minimal or no permission in the sandbox account, and corresponding CLI profiles named `Sandbox1User1` and `Sandbox1User2`. Following the best practices, create a user group called `awssecbb_iam_users` and add these users to the group.
- An S3 bucket following the *Technical requirements* section of this chapter. Create folders in the S3 bucket with the names of the IAM users we created for testing this recipe, namely `awssecbb_iam_user1` and `awssecbb_iam_user2`.

## How to do it...

In this section, we will use the AWS CLI to create an IAM policy and then attach it to a group. We will also explore the use of policy variables within this recipe. Let us get started:

1. Create a file called `s3-list-user-folder-policy.json` with the following JSON policy, but replace my bucket name with your bucket name:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:ListBucket",
      "Resource": "arn:aws:s3:::cloudericks-demo",
      "Condition": {
        "StringLike": {
          "s3:prefix": "${aws:username}/*"
        }
      }
    }
  ]
}
```

**Important note**

The policy JSON file and all the CLI commands are included with the code files.

2. Create the policy using the `create-policy` subcommand:

```
aws iam create-policy --policy-name S3ListMyFolderPolicy
--policy-document file://s3-list-user-folder-policy.json
--profile Sandbox1Admin1
```

This should return the policy's details, along with its ARN:

```
{
  "Policy": {
    "PolicyName": "S3ListMyFolderPolicy",
    "PolicyId": "ANPATAZGPUIMQUUFAFJF",
    "Arn": "arn:aws:iam::207849759248:policy/S3ListMyFolderPolicy",
    "Path": "/",
    "DefaultVersionId": "v1",
    "AttachmentCount": 0,
    "PermissionsBoundaryUsageCount": 0,
    "IsAttachable": true,
    "CreateDate": "2024-04-01T05:33:04+00:00",
    "UpdateDate": "2024-04-01T05:33:04+00:00"
  }
}
```

Figure 2.9 – The create-policy command response

3. Attach the policy to the group using the `attach-group-policy` subcommand using the policy ARN from the previous command:

```
aws iam attach-group-policy --group-name awsseccb_iam_
users --policy-arn arn:aws:iam::207849759248:policy/
S3ListMyFolderPolicy --profile Sandbox1Admin1
```

4. Verify the changes by running the `aws s3 ls` command followed by the `awsseccb_iam_user1` bucket name and folder name with the `AwsSecCbUser` profile for the `awsseccb_iam_user1` IAM user from the `awsseccb_users` group:

```
aws s3 ls cloudericks-demo/awsseccb_iam_user1/ --profile
Sandbox1User1
```

This should yield a successful response, listing all the objects from the `awsseccb_iam_user1` folder.

```
$ aws s3 ls cloudericks-demo/awsseccb_iam_user1/
--profile Sandbox1User1
2024-04-06 14:36:33          0
2024-04-06 14:37:12      58940 Screenshot 2024-04
-04 at 9.26.13 AM.png
```

Figure 2.10 – Listing the S3 folder

5. Verify the changes by running the `aws s3 ls` command followed by the `awsseccb_iam_user1` bucket name and folder name with the `Sandbox1User2` profile for the `awsseccb_iam_user2` IAM user:

```
aws s3 ls cloudericks-demo/awsseccb_iam_user1/ --profile
Sandbox1User2
```

This should yield an `AccessDenied` response as `awsseccb_iam_user2` cannot list the `awsseccb_iam_user1` folder.

```
$ aws s3 ls cloudericks-demo/awsseccb_iam_user1/
--profile Sandbox1User2

An error occurred (AccessDenied) when calling the
ListObjectsV2 operation: Access Denied
```

Figure 2.11 – Listing other user’s S3 folders

If we try to list content of the `awsseccb_iam_user2` folder with the `Sandbox1User2` profile for the `awsseccb_iam_user2` IAM user, we should get a successful response, listing all the objects from the `awsseccb_iam_user1` folder.

## How it works...

In this recipe, we created an IAM policy with the `${aws:username}` policy variable to restrict an IAM user to listing the contents of a folder within an S3 bucket only if that user’s username matches the folder name. The policy variable `${aws:username}` represents the IAM username of the user making the request. This means that if we assign the policy to a user called `awsseccb_iam_user1`, then, the `${aws:username}` policy within the IAM policy will be replaced with `${aws:username}` at runtime during policy evaluation, and the user will then get permission only to this folder.

## There’s more

AWS supports a set of predefined policy variables that can be used in IAM policies. Here’s a list of commonly used IAM policy variables:

- `${aws:username}`: This variable is replaced with the IAM username of the user making the request.
- `${aws:user-id}`: This variable represents the unique identifier of the IAM user or role making the request.
- `${aws:CurrentTime}`: The current date and time in UTC, formatted as `YYYY-MM-DDTHH:MM:SSZ`.
- `${aws:EpochTime}`: The current date and time in UTC, represented as the number of seconds since January 1, 1970 (the Unix epoch).

- `${aws:principaltype}`: The type of principal (user, account, role, federated user, etc.) making the request.
- `${aws:SecureTransport}`: A Boolean value indicating whether the request was sent using SSL.
- `${aws:SourceIp}`: The IP address of the requester. This is useful for restricting access to certain IP ranges.
- `${aws:UserAgent}`: The user agent string of the requester's client application.
- `${aws:Requester}`: The AWS account ID of the requester. Useful in cross-account scenarios.
- `${aws:sourceVpc}`, `${aws:sourceVpce}`, and `${aws:sourceVpcIpv4CidrBlock}` are variables used for controlling access based on the source VPC, VPC endpoint, or the IPv4 CIDR block of the request.
- `${aws:TagKeys}` and `${aws:RequestTag/tag-key}` are used for matching the tags on the request.
- `${aws:MultiFactorAuthPresent}`: A Boolean value indicating whether the request was made with **multi-factor authentication (MFA)**.
- `${s3:x-amz-acl}`, `${s3:x-amz-content-sha256}`, `${s3:x-amz-copy-source}`, and so on, are service-specific variables for S3, allowing policies to match various conditions specific to S3 requests.
- `${s3:prefix}` and `${s3:max-keys}` are variables specifically for controlling access based on S3 request parameters.

This list isn't exhaustive as AWS continually evolves and may introduce new variables or specific variables for different services beyond IAM. For the most current and comprehensive list, it's best to refer to the official AWS documentation.

## See also

We can read more about AWS policy variables here: <https://www.cloudericks.com/blog/understanding-aws-policy-variables-with-practical-examples>.

## Creating customer-managed policies in IAM Identity Center

To create customer-managed policies in IAM Identity Center, we need to first create a **custom permission set** and then assign users or groups to one or more AWS accounts with that permission set. In the *User management and SSO with IAM Identity Center* recipe from *Chapter 1*, we created a permission set using an AWS-managed policy. In this recipe, we will create a custom permission set based on the customer-managed IAM policy we created in the *Creating a customer-managed IAM policy* recipe in this chapter, and then, we will assign a user to an AWS account with this permission set making use of groups.

## Getting ready

We need the following to successfully complete this recipe:

- An AWS account where an IAM Identity Center instance is setup. If we are using AWS Organizations, as we saw in *Chapter 1*, this will be the **management account**. We could also do this using a **delegated administrator** account in an AWS organization. We learned about the **delegated administrator** account in *Chapter 1*. We can use the `aws-sec-cookbook-1` management account, which we set up in *Chapter 1*. Alternatively, you may follow the steps from a standalone AWS account without AWS Organizations where an IAM Identity Center instance is set up, and in that case, just use that account when I refer to `aws-sec-cookbook-1` in the recipe.

### Important note

As a good practice, we can limit access to the management account by making administrative changes using a delegated administrator as we in the *There's more...* section of the *User management and SSO with IAM Identity Center* recipe from *Chapter 1*. This is particularly advisable for larger organizations to restrict the number of people with access to the management account.

- We need a user with `AdministratorAccess` permission for the account we plan to use for this recipe, which is `aws-sec-cookbook-1` in our case. I will be using the `awsseccbadmin1` user that we created in *Chapter 1*.
- We can test the recipe using an `awsseccbuser1` user with no assignments to our account, `aws-sec-cookbook-1`. We may also assign the `awsseccbadmin1` user to another AWS account, say `awsseccb-sandbox-1`, where the user does not already have any assignments with this permission set, and test that assignment.
- We need to create a customer-managed IAM policy called `MyS3ListAllBucketsPolicy` following the *Creating a customer-managed IAM policy from the AWS Management Console* section of the *Creating a customer-managed IAM policy* recipe in this chapter, in all the accounts where we want to do the assignment. As we are planning to make an assignment in the current `aws-sec-cookbook-1` account itself, we need to create this policy within the `aws-sec-cookbook-1` account. To make an assignment in the `awsseccb-sandbox2` account, we need to create this policy within the `awsseccb-sandbox2` account. If you are following the recipes within this book, we had already created this in the `awsseccb-sandbox-1` account, and therefore, if we plan to assign the policy to that account, we do not have to create it again in that account. The policy JSON is included with the code files as `s3-list-all-my-buckets-policy.json`.
- An S3 bucket following the *Technical requirements* section of this chapter.



- To execute the CLI commands, we need to configure a CLI profile using IAM Identity Center called `AWSSECBAAdmin1` for the `awssecbadmin1` user. After the customer-managed policy is assigned to the `awssecbuser1` user for the same account, we will also need to configure a CLI profile called `AWSSECbUser1` for that user. It is important to note that CLI profiles must be configured for each user on a per-account, per-assignment basis, and it is a good practice to include the account name as part of the profile name. Therefore, if you want to assign permissions for `awssecbadmin1` to another account, say `awssecb-sandbox-1`, we can create a `Sandbox1Admin1` profile.

## How to do it...

Initially, we will create a permission set from the Management Console and utilize it for assignment. Following that, we will replicate the process using the AWS CLI.

### *Customer-managed IAM policy via the AWS Management Console*

We will first create a permission set with an existing IAM policy and then assign it to an AWS account using that permission set. Let us get started:

1. Log in to `aws-sec-cookbook-1` as the `awssecbadmin1` user and go to the **IAM Identity Center** dashboard.
2. Change the Region to `us-east-1` using the Region drop-down located in the top-right corner of the screen as we had configured the Region for IAM Identity Center as `us-east-1` in the *User management and SSO with IAM Identity Center* recipe from *Chapter 1*. If you had selected a different Region, select that Region instead.


#### **Important note**

The AWS Organizations service permits the use of IAM Identity Center in just one AWS Region at any given time. If we wish to shift the IAM Identity Center to a different Region after setting it up in one Region, we need to first delete the existing configuration in the initially chosen Region and then set it up again in the new Region.


3. Click on **Permission sets** from the left sidebar.
4. On the **Permission sets** page, click on **Create permission set**.
5. In the **Permission set type** page, select **Custom permission set** and click **Next**.
6. In the **Specify policies and permissions boundary** page, expand **Customer managed policies**, and click on **Attach policies**.

7. Enter the policy name as `MyS3ListAllBucketsPolicy`. If you gave a different name, provide that name here.

### ▼ Customer managed policies (set)

Customer managed policies are standalone policies that you create and manage in your AWS accounts to define custom permissions. You can attach up to 10 managed policies (AWS managed policies and customer managed policies) to your permission set by specifying the names of the policies exactly as they appear in your accounts. Customer managed policies are intended for advanced use cases. To ensure that you understand your shared security responsibility and best practices for configuring these policies, review the IAM Identity Center documentation. [Learn more](#) 

#### Policy names

To attach a customer managed policy to your permission set, you must specify the policy name exactly as it appears in the IAM console. To find the policy name, sign in to the [IAM console](#)  using the same AWS account as your permission set. If your permission set will be provisioned in multiple AWS accounts, a policy with the same name must exist in each account.

**Attach more**

Figure 2.12 – Selecting customer-managed IAM policy

8. Scroll down and click **Next**.
9. In the **Specify permission set details** page, give the value of **Permission set name** as `MyListAllBucketsPermission`, and **Description** as `My S3 List All Buckets Permission`, set **Session duration** as **1 hour**, leave values for other fields in the page empty, and click **Next**, which is located on the bottom-right of the page.
10. In the **Review and create** page, review everything and click on **Create**. The new permission set should now appear on the **Permission sets** page.
11. Assign the `awssecbuser1` user (or a group that contains this user) to our `aws-sec-cookbook-1` AWS account. Select the `MyListAllBucketsPermission` permission set that we created earlier in this recipe. For detailed steps to assign permission sets, we can refer to the *User management and SSO with IAM Identity Center* recipe from *Chapter 1*.
12. To verify the assignment, log into the AWS access portal as `awssecbuser1` using the AWS access portal URL of AWS Identity Center. We can get the URL from our **Identity Center** dashboard. It is also present in the invitation email sent to the user's email when the user was created. We should now be able to see the `aws-sec-cookbook-1` account (or any other accounts we assigned to) after we log into the access portal.

13. Log into the `aws-sec-cookbook-1` account from the access portal and go to the **Amazon S3** service.
14. Click on **Buckets** from the left sidebar. We should be able to see all the buckets available in the account.

We will now create a permission set and assign it to the AWS account using CLI. However, before moving forward, we should remove the assignment we made in this section as we will be doing the same assignment from the AWS CLI.

### ***Customer-managed IAM policy via the AWS CLI***

In this section, we will first create a permission set and then assign a group that contains our user to an AWS account using that permission set. Let us get started:

1. Configure CLI for the `awssecadmin1` user for the `aws-sec-cookbook-1` account with the `AwsSecCbAdmin` profile name, and log in using that profile following the *User management and SSO with IAM Identity Center* recipe from *Chapter 1*.

#### **Important note**

As we learned in *Chapter 1*, we can use the `aws configure sso` command, which is primarily used to set up a profile initially using IAM Identity Center, which provides values for `sso_start_url` and `sso_region`. This configuration is typically a one-time activity unless we need to change or update the profile's settings. After that, we can utilize the `aws sso login` command each time we want to start a session and use the `aws sso logout` command to log out from that session.

2. Create a permission set using the `create-permission-set` command:

```
aws sso-admin create-permission-set \
  --instance-arn <Your-SSO-Instance-ARN> \
  --name MyS3ListAllBucketsPermissionCLI \
  --description "S3 List All Buckets Permission" \
  --session-duration "PT1H" \
  --profile AwsSecCbAdmin
```

We can get the value of `<Your-SSO-Instance-ARN>` from the **Settings** page within IAM Identity Center in the AWS Management Console.

If the permission set is created successfully, we should get a response with the new permission set ARN like the following:

```
heartin$aws sso-admin create-permission-set \
  --instance-arn arn:aws:sso::instance/ssoins-7223c406a1331fac \
  --name MyS3ListAllBucketsPermissionCLI \
  --description "S3 List All Buckets Permission" \
  --session-duration "PT1H" \
  --profile AwsSecCbAdmin
{
  "PermissionSet": {
    "Name": "MyS3ListAllBucketsPermissionCLI",
    "PermissionSetArn": "arn:aws:sso::permissionSet/ssoins-7223c406a1331fac/ps-fbf56a222e68cf94",
    "Description": "S3 List All Buckets Permission",
    "CreateDate": "2023-09-13T00:09:25.950000+05:30",
    "SessionDuration": "PT1H"
  }
}
heartin$
```

Figure 2.13 – Request and response for the create-permission-set subcommand

3. Attach our customer-managed policy to the permission set using the `attach-customer-managed-policy-reference-to-permission-set` subcommand:

```
aws sso-admin attach-customer-managed-policy-reference-to-permission-set \
  --instance-arn <Your-SSO-Instance-ARN> \
  --permission-set-arn <Your-permission-set-ARN> \
  --customer-managed-policy-reference
Name=MyS3ListAllBucketsPolicy \
  --profile AwsSecCbAdmin
```

For `<Your-permission-set-ARN>`, we need to use the permission set ARN we received from the previous command. This command gives us no response.

4. Assign the permission set to our AWS account and group using the `create-account-assignment` subcommand:

```
aws sso-admin create-account-assignment \
  --instance-arn <Your-SSO-Instance-ID> \
  --permission-set-arn <Your-Permission-Set-ARN> \
  --target-id <Target-Account-ID> \
  --target-type AWS_ACCOUNT \
  --principal-id <Your-Group-ID> \
  --principal-type GROUP
```

<Target-Account-ID> is the account ID of the account to which we want to make an assignment and <Your-Group-ID> is the ID of the group (not name), and both can be found within IAM Identity Center in the AWS Management Console. This command should immediately give a response with Status as IN\_PROGRESS:

```
{
  "AccountAssignmentCreationStatus": {
    "Status": "IN_PROGRESS",
    "RequestId": "12693ec2-e406-426f-b40e-cf5eabde7162",
    "TargetId": "207849759248",
    "TargetType": "AWS_ACCOUNT",
    "PermissionSetArn": "arn:aws:sso::permissionSet/ssoins-7223c406a1331fac/ps-fbf56a222e68cf94",
    "PrincipalType": "GROUP",
    "PrincipalId": "2438c478-7071-7098-2a92-3bcfe92164f5"
  }
}
```

Figure 2.14 – Response for the create-account-assignment subcommand

5. Verify the assignment using the list-account-assignments subcommand:

```
aws sso-admin list-account-assignments \
  --instance-arn <Your-SSO-Instance-ARN> \
  --account-id <Assigned-Account-ID> \
  --permission-set-arn <Your-Permission-Set-ARN> \
  --profile AwsSecCbAdmin
```

This should give us a response with the assignment details:

```
{
  "AccountAssignments": [
    {
      "AccountId": "207849759248",
      "PermissionSetArn": "arn:aws:sso::permissionSet/ssoins-7223c406a1331fac/ps-fbf56a222e68cf94",
      "PrincipalType": "GROUP",
      "PrincipalId": "2438c478-7071-7098-2a92-3bcfe92164f5"
    }
  ]
}
```

Figure 2.15 – Response of the list-account-assignments subcommand

6. Configure a CLI profile called AwsSecCbUser1 for the awsseccbuser1 user with a profile name of AwsSecCbUser1 and log in using that profile following the *User management and SSO with IAM Identity Center* recipe from *Chapter 1*.

7. Next, verify access by running the `aws s3 ls` command:

```
aws s3 ls --profile AwsSecCbUser1
```

This should list all the S3 buckets that `awsseccb_user1` has permissions to view, in line with `MyS3ListAllBucketsPolicy`:

```
heartin$aws s3 ls --profile AwsSecCbUser1
2023-08-13 16:38:07 awsseccb
```

Figure 2.16 – List all buckets after successful login

We completed the recipe from the management account of our AWS organization. We may also do this from a member account designated as a delegated administrator.

## How it works...

In this recipe, we learned to create permission sets based on a customer-managed policy that we had already created in the *Creating a customer-managed IAM policy* recipe from this chapter. In *Chapter 1*, we delved into the AWS IAM Identity Center service and explored creating permission sets based on an AWS-managed policy.

We set **Region** as `us-east-1`, the same Region we configured when IAM Identity Center was created. The AWS Organizations service permits the use of IAM Identity Center in just one AWS Region at any given time. If we wish to shift IAM Identity Center to a different Region after setting it up in one, we need to first delete its existing configuration in the initially chosen Region.

In the AWS Management Console, we attached the customer-managed policy to the permission set during the permission set creation by specifying the name of the policy. In the command line, it was a two-step process. First, we created the permission set and then attached the customer-managed policy to the permission set. A policy with an identical name needs to be available in the member account where this permission set is assigned. The permission set can be created from the management account, or a member account designated as the IAM Identity Center delegated administrator. The customer-managed policy must reside in the member account receiving the permission set, not in the management or administrator account.

In the command line section of the recipe, we used two AWS CLI v2 command namespaces related to IAM Identity Center: `sso-admin` and `sso`. We first used the `aws configure sso` command to configure a CLI profile with values such as `sso_start_url` and `sso_region`. This configuration is typically a one-time activity unless you need to change or update the profile's settings. After that, we used the `aws sso login` command, which we can use every time we want to log in.

The commands in the `sso-admin` namespace help administrators manage the IAM Identity Center settings, such as assigning user groups to AWS accounts with specific permission sets or listing IAM Identity Center instances. On the other hand, the commands in the `sso` namespace are designed for the end-user experience such as logging in and out of IAM Identity Center-enabled AWS accounts.

The `sso` namespace commands authenticate through IAM Identity Center, making use of temporary AWS credentials without the need to enter the username and password again, thus making access to AWS services more secure by eliminating the need for managing multiple credentials.

## There's more...

Although the AWS SSO service has been rebranded as AWS IAM Identity Center in the AWS Management Console, the CLI namespaces such as `sso-admin` and `aws sso` remain unchanged to maintain backward compatibility.

## See also

- We can find more details about the `sso-admin` namespace and its commands at <https://awscli.amazonaws.com/v2/documentation/api/latest/reference/sso-admin/index.html>.
- We can find more details about the `sso` namespace and its commands at <https://awscli.amazonaws.com/v2/documentation/api/latest/reference/sso/index.html>.

## Setting IAM permission boundaries for IAM entities

This recipe demonstrates how to use **permissions boundaries** to set a maximum permission limit for an IAM entity such as an IAM user or an IAM role. Initially, we will assign a user complete access to S3. Subsequently, we'll apply a permission boundary to confine the user's S3 permissions exclusively to read-only access. Similar to SCPs, permission boundaries do not grant permissions; they only define constraints. In other words, without an identity-based, resource-based, or session policy in place, actions permitted within the permission boundary or SCP cannot be executed.

## Getting ready

We need the following to successfully complete this recipe:

- A working AWS account, `awsseccb-sandbox-1`, and a user with `AdministratorAccess` permission to that account, `awsseccbadmin1`, following the *Technical requirements* section of this chapter.
- An IAM user, `awsseccb_iam_user1`, with `AmazonS3FullAccess` permission to the account, `awsseccb-sandbox-1`.
- Understand how to create an Amazon S3 bucket.

## How to do it...

We can create a permission set as follows:

1. Log into the AWS Management Console of `awssecb-sandbox-1` as `awssecbadmin1` and go to the **IAM** dashboard.
2. Click on **Users** in the left sidebar and click on the name of the user to use for this recipe, which is `awssecb_iam_user1` in our case.
3. Scroll down to the **Permissions boundary** section and click on **Set permissions boundary**.

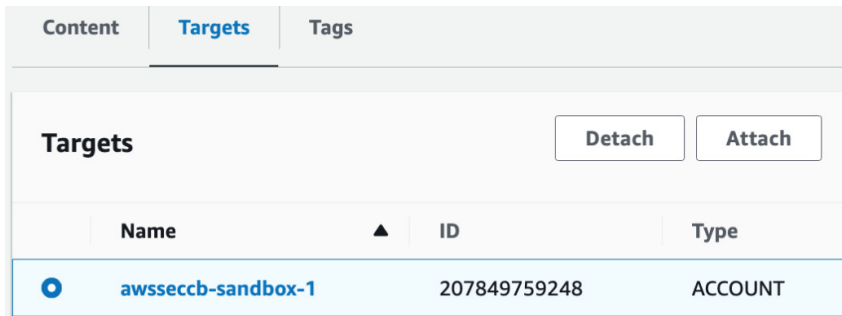


Figure 2.17 – Set permission boundary

4. Search for **AmazonS3ReadOnlyAccess**, select it, and click on **Set boundary**.
5. Log into the AWS Management Console of our account as the `awssecb_iam_user1` user, go to the S3 service dashboard, and try to create a bucket with the default configuration. We should get an error message such as the following even if the user has `AmazonS3FullPermission`.

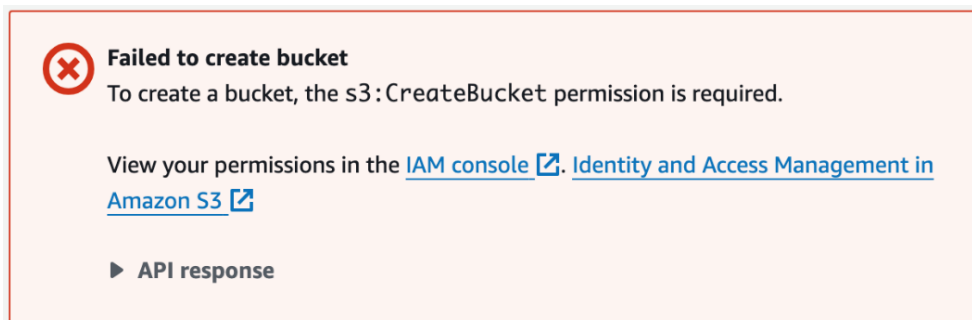



Figure 2.18 – Failed to create bucket

6. Log in to the account as `awssecbadmin1`, go to the `awssecb_iam_user1` **Permissions boundary** section, and click on **Remove boundary**, as shown in the following figure. In the confirmation dialog popup, click on **Remove boundary**.



### ▼ Permissions boundary (set)

Set a permissions boundary to control the maximum permissions for this user. Use this advanced feature used to delegate permission management to others. [Learn more about permission boundaries](#) 

Permissions boundary

 [AmazonS3ReadOnlyAccess](#)  AWS managed

Change boundary

Remove boundary

Figure 2.19 – Remove permissions boundary

7. Log in to the AWS Management Console of our account as the `awssecb_iam_user1` user, go to the S3 service dashboard, and try to create a bucket with the default configuration. We should be able to successfully create the bucket now.
8. Log in to the Management Console as `awssecbadmin1`, go to the **IAM** dashboard, and remove all existing permissions, directly assigned, or inherited. For inherited permissions from groups, this can be done by removing the user from all groups from the **User groups membership** section under the **Groups** tab. From the **Users** pane, click on the name of our `awssecb_iam_user1` user, and go to the **Groups** tab. Directly assigned permissions can be removed from the **Permissions** tab. Verify that no permissions policies are attached to the user in the **Permissions** tab.
9. Go to the **Permissions boundary** section and click on **Set permissions boundary**, search for `AmazonS3FullAccess`, select it, and click on **Set boundary**.
10. Log into the AWS Management Console of our account as the `awssecb_iam_user1` user, go to the S3 service dashboard, and try to create a bucket with the default configuration. We should get an error message like the one we got in *Step 5*. This demonstrates that even if we allow permissions within the permission set, without an identity-based, resource-based, or session policy in place, actions permitted within the permission boundary are not allowed.

You may remove the permissions boundary before proceeding with further recipes.

## How it works...

In AWS IAM, permission boundaries serve as a control mechanism to set the maximum permissions a user or group can have, regardless of any policies directly attached to them. By attaching a permission boundary to a user, we define the upper limit of their permissions. In the scenario described, the demo user initially inherits full S3 access permissions. However, when a permission boundary policy, such as

`AmazonS3ReadOnlyAccess`, is applied to the user, it restricts their S3 permissions to read-only access. When permission boundaries, SCPs, and identity-based policies are concurrently applied, an action is authorized only if it is explicitly allowed by all three components. In situations where only SCPs and a permissions boundary are in place without any identity-based or resource-based policies authorizing an action, the default outcome is denial.

## There's more...

In situations where only SCPs and a permissions boundary are in place without any identity-based or resource-based policies authorizing an action, the default outcome is denial.

## See also

We can read more about permissions boundaries for IAM entities at <https://www.cloudericks.com/blog/getting-started-with-permissions-boundaries-in-aws>.

# Centralizing governance in AWS Organizations with SCPs

**SCPs** in AWS allow us to manage permissions across an entire AWS organization, **organizational units (OUs)**, or even individual accounts. SCPs fulfill the crucial need for centralized governance by allowing administrators to implement consistent compliance and security policies efficiently across multiple AWS accounts. By utilizing SCPs, organizations can enhance their security posture, manage risks more effectively, and ensure adherence to both internal policies and external regulatory requirements through a centralized policy management framework. In this recipe, we will use an SCP to restrict the creation of Amazon S3 buckets within a specific Region.

## Getting ready

We need a working AWS account with AWS Organizations service enabled. I will be using the `aws-sec-cookbook-1` account that we created in *Chapter 1*.

## How to do it...

We can explore SCPs as follows:

1. Log in to the AWS Management Console and navigate to the **AWS Organizations** service.
2. Click on **Policies** in the left sidebar.

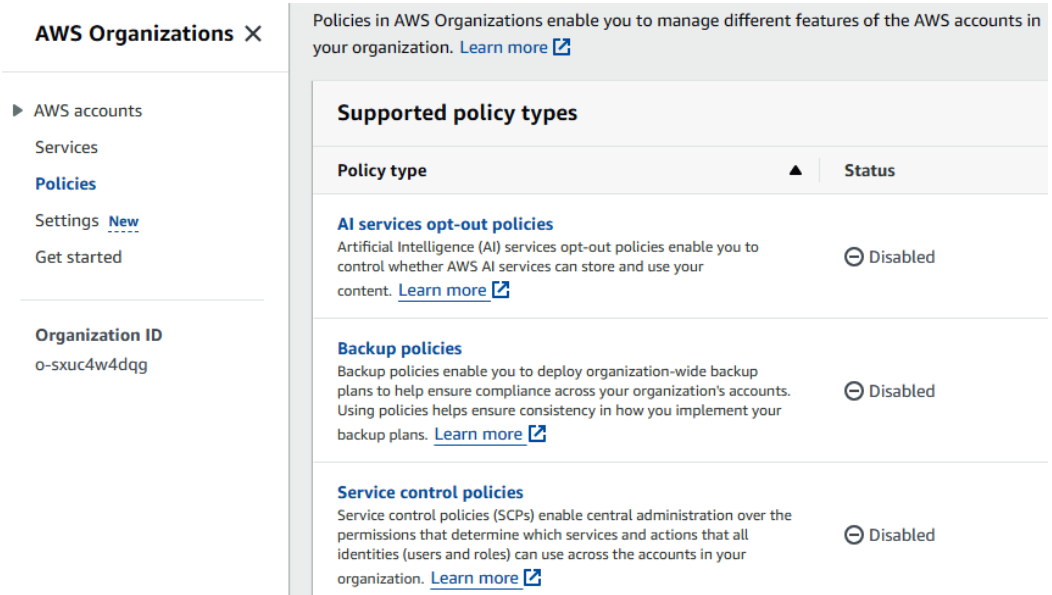


Figure 2.20 – Policies dashboard

3. Click on the **Service control policies**. If we are using SCPs for the first time, then click on **Enable service control policies**. In the **Service control policies** pane, we can see that a policy called FullAWSAccess is already present.
4. In the **Service control policies** pane, click on **Create policy**.
5. Enter SandboxS3BucketCreate for the **Policy name** field and optionally enter Sandbox S3 Bucket Create Policy for the **Policy description - optional** field.
6. Paste the given policy in the policy section and then scroll down and click on **Create policy**. The policy allows S3 bucket creation in any Region except us-east-1.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Statement1",
      "Effect": "Deny",
      "Action": [
        "s3:CreateBucket"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
```

```

    "StringEquals": {
      "aws:RequestedRegion": "us-east-1"
    }
  }
}
]
}

```

7. In the **Service control policies** pane, select the policy we created, click on the **Actions** dropdown, and then click on **Attach policy**.
8. Select the OU or account to which we want to attach the policy and then click on **Attach policy**. I have selected the `awsseccb-sandbox-1` account that I created in *Chapter 1*. We can select multiple OUs and accounts as needed.
9. Log in to the account we attached the policy to, which is `awsseccb-sandbox-1` in my case, as a user with `AdministratorAccess` permission, and try to create an S3 bucket in the `us-east-1` Region. We should get an error message that we do not have the required permission.

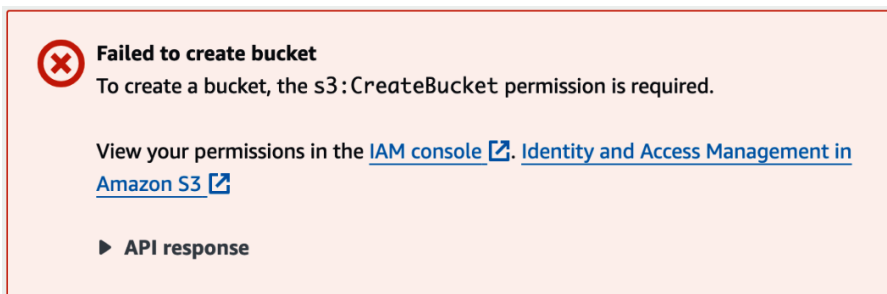


Figure 2.21 – Insufficient permissions error

10. Create an S3 bucket in another Region, for instance, `us-east-2`, and it should be successful.
11. To detach the policy, we can log in to the management account, navigate to the service control policy we created, go to the **Targets** tab, select the account, and click on **Detach**.



Figure 2.22 – The Targets tab to detach policy from the account

12. Once you detach the policy, navigate back to our test account, `awsseccb-sandbox-1` in my case, and try to create an S3 bucket in the `us-east-1` Region. This time we should be able to create the bucket.

## How it works...

We used an SCP to restrict S3 bucket creation in a specific Region by defining a `Deny` statement within the SCP targeting the action of creating S3 buckets (`s3:CreateBucket`) and associating it with a condition that checks whether the requested Region is the one to be restricted (e.g., `us-east-1`). This approach ensures centralized control and governance over S3 bucket deployments, enabling organizations to enforce regional compliance standards, enhance security posture, and optimize resource utilization within their AWS environment.

SCPs are disabled by default, but we can enable them. SCPs, if enabled, need to explicitly allow services and actions that are allowed. When we enable SCPs, AWS attaches an SCP called `FullAWSAccess` to all accounts and OUs within that organization. This SCP allows every action and service. We can then create additional SCPs to deny services or actions and `Deny` always takes precedence. This is the default strategy. With this strategy, we do not have to do anything when new services or actions are introduced by AWS as they are allowed by the `FullAWSAccess` SCP.

We may also replace the `FullAWSAccess` SCP with another SCP that allows only a specific set of services and features. If we do this, we must explicitly add any new services or actions that AWS introduces later to this SCP. However, if we follow the strategy discussed in the previous point, which is having the `FullAWSAccess` SCP and adding another SCP with the list of services and actions to deny, we do not have to do anything when new services or actions are introduced by AWS.

If SCPs are enabled, every account, root OU, and other OUs need to have at least one SCP attached. Therefore, if we need to replace the `FullAWSAccess` SCP with another, we need to first create and attach another SCP, and then detach the `FullAWSAccess` SCP.

## There's more...

SCPs, similar to permission boundaries in IAM, define the maximum permissions an entity may hold but do not grant permissions on their own. Therefore, when permission boundaries, SCPs, and identity-based policies are concurrently applied, an action is authorized only if it's explicitly allowed by all three components. Additionally, in situations where only SCPs and a permissions boundary are in place without any identity-based or resource-based policies authorizing an action, the default outcome is denial. This underlines the principle that permissions must be explicitly granted, and in the absence of such explicit permissions, access is not permitted. We discussed policy types earlier in this chapter.

## See also

We can read more about service control policy examples here: <https://www.cloudericks.com/blog/understanding-aws-scps-and-the-deny-list-and-allow-list-strategies>.

## IAM cross-account role switching and identity account architecture

Many organizations use multiple AWS accounts to distinctly manage different operational environments, such as development, testing, and production. Users with varied job roles may need diverse access privileges across these environments. However, the task of managing multiple IAM users, each having different access credentials across various AWS accounts, can be complex and time consuming.

A role in AWS IAM grants a specific set of permissions, much like a user account. Unlike users, we do not directly log in to roles; instead, we can switch to a role either in our own account or another AWS account. This replaces our original permissions with the role's permissions. Based on how the role switch happens, two primary strategies are often utilized to streamline user access management across multiple AWS accounts, negating the need for individual IAM users and access credentials for each account.

The first approach for role switching involves SSO, where users authenticate via a central access portal and are granted access to multiple accounts, each with distinct roles and permissions. In this model, the role switching happens automatically. The second approach for role switching leverages the identity account architecture where users log in to an identity or central account and use the AWS switch role feature to access resources across various accounts, each assigned specific roles and permissions.

After AWS rebranded AWS SSO as IAM Identity Center, AWS recommends using the IAM Identity Center service as a best practice over using IAM directly. Therefore, the first approach using SSO with IAM Identity Center is preferred today for managing multiple accounts. However, many, especially smaller, organizations, who do not want the overhead of maintaining the AWS Organizations service, still use the switch role feature extensively. We can use the switch role feature in conjunction with IAM Identity Center, and this compatibility facilitates a seamless shift from the traditional approach of using IAM users with the identity account architecture to employing IAM Identity Center.

In this recipe, we will learn to implement the second approach that uses the identity account architecture, making use of the switch role feature that will allow the switching of roles from a source account to a destination account. The first approach of using IAM Identity Center (previously known as AWS SSO) was explored in the recipes in *Chapter 1* and the earlier recipes in this chapter.

### Important note

Switching roles can be performed by an IAM user, a user in the IAM Identity Center, a SAML-federated role, or even a web-identity federated role. However, switching roles is not permissible for the AWS account root user.

In AWS Organizations, an all-access role featuring a trust policy is automatically established in member accounts. This role facilitates the management account in switching roles to any member account created within AWS Organizations. Therefore, administrators intending to switch roles between a

management and a member account in AWS Organizations can skip the setting up sections in this recipe. They can directly proceed to the sections on switching roles, utilizing the role configured during the AWS Organizations setup, as demonstrated in the *Multi-account management with AWS Organizations* recipe from *Chapter 1*.

## Getting ready

We need the following to successfully complete this recipe:

- Two AWS accounts, one as the source account and the other as the destination account. These could be standalone accounts or ones that are part of an AWS organization. We can use the two member accounts we created in the *Multi-account management with AWS Organizations* recipe from *Chapter 1*, namely, `awsseccb-sandbox-1` as the source account and `awsseccb-sandbox-2` as the destination account.
- A user with administrative permissions is required in both the source and destination accounts to create roles and policies. Using IAM Identity Center, we can create a single user, `awsseccbadm1n1`, and grant it `AdministratorAccess` permission to both accounts. If opting for traditional IAM, a separate user must be created in each account with this permission.
- An IAM user named `awsseccb_iam_user1` in the source account, `awsseccb-sandbox-1` in our case, with no permission or with the `IAMUserChangePassword` permission if you selected the **Users must create a new password at next sign-in - Recommended** option. We could also use IAM Identity Center. However, using IAM Identity Center may be considered redundant here as we can use the access portal with IAM Identity Center to log into multiple AWS accounts without the need to use the account switching functionality. Nevertheless, we can do that and to understand the concepts better, you may try that option as well.
- An S3 bucket following the *Technical requirements* section of this chapter.
- To execute the steps in sections related to the AWS CLI, we need to set up two AWS CLI profiles for the `awsseccbadm1n1` admin user. First, `AWSSECAdmin1D` for the destination account and second, `AWSSECAdmin1S` for the source account. Additionally, we need a profile named `AWSSECUser1S` for the `awsseccbuser1` user in the source account. With regular IAM, users need AWS CLI profiles per user per account. With IAM Identity Center, users need AWS CLI profile per user per role (`PermissionSet`) per account.

## How to do it...

To switch roles, the destination account must have an assignable role with a trust policy that enables the source account to assume this role. Additionally, the source account needs a policy that authorizes this switch. We will first configure the destination and the source account from the AWS Management Console. We will also achieve this using the AWS CLI.

## Setting up a destination account via AWS Management Console

In the destination account, we need a role along with a trust policy that allows the source account to assume this role. We will create a role with an existing permissions policy and specify the source account ID for the trust relationship. Let us get started:

1. Log into the AWS Management Console of the destination account, say `awssecb-sandbox-2`, as the `awssecbadmin1` admin user.
2. Go to the **IAM** dashboard.
3. Click on **Roles** from the left sidebar.
4. Click on **Create role**.
5. In the **Select trusted entity** page, under **Trusted entity type**, select **AWS account** as shown in the following figure.

**Select trusted entity** [Info](#)

**Trusted entity type**

- ☐ **AWS service**  
Allow AWS services like EC2, Lambda, or others to perform actions in this account.
- ☒ **AWS account**  
Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.
- ☐ **Web identity**  
Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.
- ☐ **SAML 2.0 federation**  
Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.
- ☐ **Custom trust policy**  
Create a custom trust policy to enable others to perform actions in this account.

Figure 2.23 – Selecting trusted entity type for a role

6. Scroll down, select **Another AWS Account**, enter the 12-digit **Account ID** of the source account (in my case, it will be the ID of the `awssecb-sandbox-1` account), and click **Next** in the bottom-right corner of the page.



**An AWS account**  
Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.

☐ This account (206722961012)  
☒ **Another AWS account**

**Account ID**  
Identifier of the account that can use this role

207849759248

Account ID is a 12-digit number.

**Options**

☐ Require external ID (Best practice when a third party will assume this role)

☐ Require MFA  
Requires that the assuming entity use multi-factor authentication.

Cancel Next

Figure 2.24 – Configuring Account ID of source account

7. In the **Add permissions** page, search for S3, select `AmazonS3ReadOnlyAccess`, and click **Next** in the bottom-right corner of the page.
8. In the **Name, review, and create** page, enter `SA-S3ReadOnly` in the **Role name** field and `Amazon S3 Read Only Access Role` in the **Description** field. I have given a prefix of SA to denote that this is a switch account role.
9. Review the role details and click **Create role** in the bottom-right corner of the page.
10. Once we see a success message that says **Role SA-S3ReadOnly created**, click on **View role**, make note of the role's ARN, and select **Link to switch roles in console**.

Next, we will create an IAM policy in the source account to grant users in the source account the ability to assume the newly created role.

### ***Setting up source account via the AWS Management Console***

When administrators in the source account are switching roles, only the creation of the role with a trust policy in the destination account is necessary. There is no requirement for a policy creation or attachment in the source account, as administrators already have the `sts:AssumeRole` permission. Therefore, if we are an administrator in the source account, we can skip the setting up section for the source account in this recipe.

We can create the required IAM policy in the source account as follows:

1. Log in to the AWS Management Console of the source account (awssecb-sandbox-1 in my case), as the awssecbadmin1 admin user and go to the **IAM** dashboard.
2. Click on **Policies** from the left sidebar.
3. Click on **Create policy**.
4. Click on the **JSON** tab and then paste the following policy JSON in the **policy editor**. Remember to replace `arn:aws:iam::DESTINATION_ACCOUNT_ID:role/ROLE_NAME` with the role's ARN that we noted in the previous section. Alternatively, we can just replace `DESTINATION_ACCOUNT_ID` with the destination account's ID and `ROLE_NAME` with the role name we created in the previous section:

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "sts:AssumeRole",
    "Resource": "arn:aws:iam::DESTINATION_ACCOUNT_ID:role/
ROLE_NAME"
  }
}
```

5. Click **Next** in the bottom-right corner of the page.
6. In the **Review and create** page, enter `AR_Sandbox2_S3ReadOnly` in the **Policy name** field, and enter `AssumeRole policy for S3ReadOnlyAccess in awssecb-sandbox-2` in the **Description** field.
7. Review the details and click **Create policy** in the bottom-right corner of the page. We should get a **Policy AR\_Sandbox2\_S3ReadOnly created** message.
8. Attach this policy to the `awssecbusers` group in the source account that has the `awssecbuser1` user. With IAM, we can directly attach the policy to the group in the source account. With IAM Identity Center, we must first create a permission set with this policy and assign it to the group during the assignment of the source account as we saw in *Chapter 1*.

Next, we will switch roles from the source account to the destination account.

### **Switching roles via the AWS Management Console**

For switching roles from the source account to the destination account, follow these steps:

1. Log into the AWS Management Console of the source account, say `awssecb-sandbox-1`, as the user who was assigned the permission to switch roles, say `awssecb_iam_user1`. If we are logging in via the AWS IAM Identity Center access portal, we need to make sure we select the role for the source account with the `AssumeRole` policy.

2. Click on the drop-down menu next to the username and click on **Switch role** as shown in the following figure:

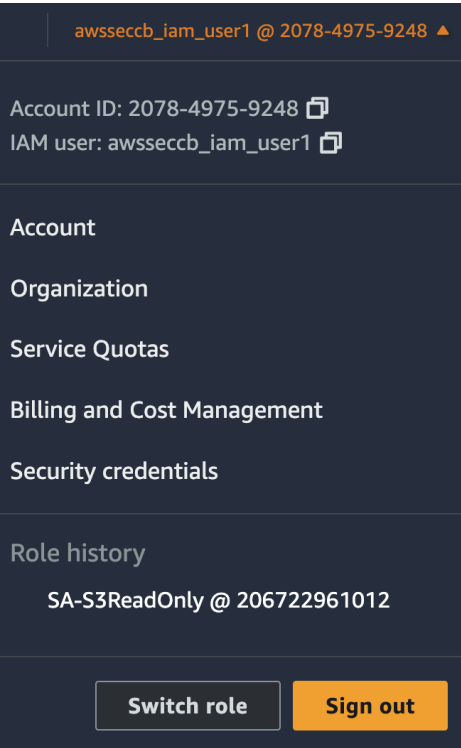



Figure 2.25 – User menu with the Switch role button

Here, I am switching roles for the second time to show the **Role history** details. On your first attempt, you will not see the **Role history** details.

3. On the **Switch Role** screen, for **Account**, enter the account number of the destination account and for **IAM role name**, enter SA-S3ReadOnly.

## Switch Role

Switching roles enables you to manage resources across Amazon Web Services accounts using a single user. When you switch roles, you temporarily take on the permissions assigned to the new role. When you exit the role, you give up those permissions and get your original permissions back. [Learn more](#) 

### Account ID

The 12-digit account number or the alias of the account in which the role exists.

### IAM role name

The name of the role that you want to assume. You can get this from the end of the role's ARN.

For example, ARN: `arn:aws:iam::111111111111:role/RoleName`

### Display name - *optional*

This name will appear in the console navigation bar when active. Choose a name to help identify the permission set assigned to the role.

### Display color - *optional*

The selected color displays in the console navigation when this role is active

Figure 2.26 – The Switch Role screen

Once we are logged in as the required user (here, `awssecbcb_iam_user1`) or the required source account (here, `awssecbcb-sandbox-1`), we can also switch roles by pasting the link to switch roles in the console that we got in *Step 10* in the *Setting up destination account via AWS Management Console* section.

4. Click on **Switch Role**.

5. This time, we should be logged in to the new account. We can verify this from the drop-down menu next to our account name in the taskbar:

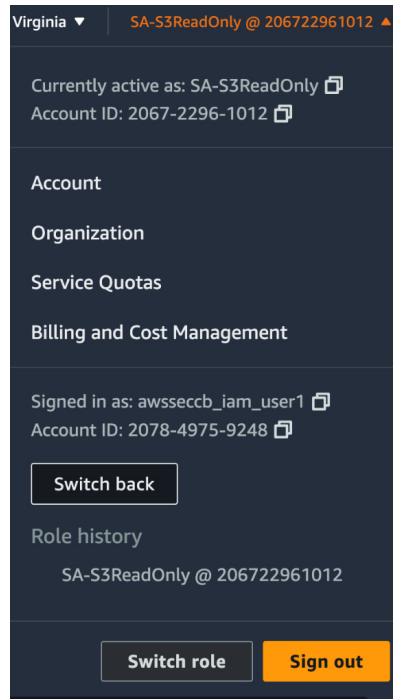


Figure 2.27 – User menu after switching role

6. Go to the **S3** service and verify that we can see all buckets and their details.
7. We can switch back to the parent account by clicking on the **Switch back** button, as we can see in *Figure 2.27*.

Next, we will see how we can create a role from the command line.

### ***Setting up a destination account via the AWS CLI***

With the AWS CLI, first, we need to create a role embedded with a trust policy in the destination account and then associate the permissions policy with this role. We will run the CLI commands using the `AWSecCBAdmin1D` CLI profile for the `awssecbadmin1` user, which is configured for the destination account (`awssecb-sandbox-2` in my case) with `AdministratorAccess` permission. Let us get started:

1. First, we need to create a **trust policy** allowing the destination account to trust the source account and save the file as `trust-policy.json`:

```
{
  "Version": "2012-10-17",
```

```

    "Statement": [
      {
        "Effect": "Allow",
        "Principal": {
          "AWS": "arn:aws:iam::<SOURCE_ACCOUNT_ID>:root"
        },
        "Action": "sts:AssumeRole"
      }
    ]
  }
}

```

Replace <SOURCE\_ACCOUNT\_ID> with your source account's 12-digit account ID.

2. Create a role using the trust relationship file created in the previous step:

```

aws iam create-role \
  --role-name SA-S3ReadOnlyRoleCLI \
  --assume-role-policy-document file://trust-policy.json \
  --profile AWSecCBAdmin1D

```

If successful, the command will return an output with the details of the newly created role including **RoleId**, and role's **ARN**.

3. Attach the AmazonS3ReadOnlyAccess permission policy to the newly created role:

```

aws iam attach-role-policy \
  --role-name SA-S3ReadOnlyRoleCLI \
  --policy-arn arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess \
  --profile AWSecCBAdmin1D

```

There is no output for this command.

Next, we need to create a policy in the source account that authorizes the role switch.

### Setting up a source account via the AWS CLI

We will run the CLI commands using the AWSecCBAdmin1S CLI profile for the awsseccbadmin1 user, which is configured for the source account (awsseccb-sandbox-1 in my case) with AdministratorAccess permission. Let us get started:

1. Create a policy to allow source account identities to assume the role in the destination account and save the file as assume-role-policy-cli.json:

```

{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",

```

```

    "Action": "sts:AssumeRole",
    "Resource": "arn:aws:iam::DESTINATION_ACCOUNT_ID:role/
SA-S3ReadOnlyRoleCLI"
  }
}

```

Replace `DESTINATION_ACCOUNT_ID` with the destination account's 12-digit Account ID.

2. Create a policy that allows assuming the `S3ReadOnlyRoleCLI` role in the destination account using the `assume-role-policy-cli.json` file we created in *Step 1*:

```

aws iam create-policy \
  --policy-name AR_Sandbox2_S3ReadOnly_CLI \
  --policy-document file://assume-role-policy-cli.json \
  --profile AWSecCBAdmin1S

```

3. Attach this policy to the `awsseccbusers` group in the source account, which has the `awsseccbuser1` user. With IAM, we can directly attach the policy to the group in the source account. With IAM Identity Center, we must first create a permission set with this policy and assign it to the group during the assignment to the source account as we saw in *Chapter 1*.

Next, we will switch roles from the source account to the destination account.

### Switching roles via the CLI

We need to use the `AWSecCBUser1S` CLI profile for the `awsseccbuser1` user, configured for the source account for the role with the `sts:AssumeRole` permission. Let us get started:

1. Establish a new profile for the role within the `.aws/config` file. On Unix or Linux systems, this file is in the user's home directory. For Windows users, we can find the file at `C:\Users\USERNAME\.aws\config`. Make sure to replace `USERNAME` with your actual Windows username:

```

[profile switchrole]
role_arn = arn:aws:iam::DESTINATION_ACCOUNT_ID:role/
SA-S3ReadOnlyRoleCLI
source_profile = AWSecCBUser1S

```

We need to replace the role's ARN with the ARN of the role we created in the destination account. Whether we want to do a cross-account role assumption or role assumption within the same account, we need to follow this step.

2. Run the `aws s3 ls` command with this new user profile as follows:

```
aws s3 ls --profile switchrole
```

This should give us a successful response with the names of the buckets in the destination account.

**Important note**

To obtain the role's credentials, the AWS CLI internally uses `sts:AssumeRole` for assuming the role utilizing the credentials associated with the `source_profile`, which is `AWSSECUser1S` in our case. Therefore, the identity linked to `source_profile` should possess the `sts:AssumeRole` permission for the role mentioned in `role_arn`.

3. We can also verify the switch role changes within this recipe by running the `aws sts assume-role` command:

```
aws sts assume-role \
  --role-arn ROLE_ARN \
  --role-session-name SESSION_NAME \
  --profile AWSSECUser1S
```

The `aws sts assume-role` command, if successful, should give us `AccessKeyId`, `SecretAccessKey`, and `SessionToken`. We can use the credentials returned from the `sts assume-role` command through APIs, including from the CLI.

## How it works...

In this recipe, we configured two accounts, a source account and a destination account, such that we can switch roles from the source account and log in to the destination account, assuming there's a role in the destination. This feature is used in an identity account architecture where all IAM users are managed in a single AWS account, often referred to as the *identity account*, and can switch roles to access resources across multiple accounts, thus eliminating the need for distinct login credentials for each account.

To switch roles between accounts, the account we are switching to must have a role set up for this purpose. The role in the destination account should include a trust policy that permits the source account to assume this role, and the source account must have a policy with the `sts:AssumeRole` permission, which allows this role assumption. In summary, to switch roles between two AWS accounts – let us call them the source and destination accounts – the following configurations are needed:

- First, a role that can be assumed by an identity (user or service) in the source account must exist in the destination account.
- Second, the identity in the source account must be granted the `sts:AssumeRole` permission. This permission should specify the ARN of the role to be assumed in its policy. This step is needed for role-switching within a single account or between two different accounts.
- For **cross-account role switching**, as seen in this recipe, the destination account must also have a trust policy attached to the role to allow the source account to assume that role.



The trust policy attached to the role in the destination account is an example of a resource-based policy, while the policy within the source account authorizing the role switch is an example of an identity-based policy. Implicitly, session policies are used within the recipe. IAM policies attached to the role in the destination account serve as session policies. When a role is assumed, whether within the same account or across accounts, AWS **Security Token Service (STS)** generates temporary security credentials for the session. These credentials include session policies, defining the permissions available within the temporary session.

Once a role is assumed, the permissions granted are determined solely by the session policies. Additionally, any permissions associated with the IAM identity assuming the role are temporarily relinquished, ensuring that the temporary session operates strictly within the boundaries of the assumed role's permissions, mitigating the risk of unauthorized access. Furthermore, explicit use of the `aws sts assume-role` command allows for the obtaining of temporary security credentials. Upon success, this command provides `AccessKeyId`, `SecretAccessKey`, and `SessionToken`. These credentials can be used through APIs, including the AWS CLI, enabling controlled access to AWS resources based on the permissions defined by the assumed role's session policies.

When we set up a new account within AWS Organizations, AWS automatically creates an IAM role in the new member account. This role is designed to allow administrators in the management account to assume the role and access the member accounts. This role is commonly named `OrganizationAccountAccessRole` but we can change it to a different one while creating an account under an AWS organization, as we saw in *Chapter 1*. If we are using accounts that are not part of AWS Organizations, we need to create these manually.

If we are using AWS IAM Identity Center with AWS Organizations, we can easily log in to different AWS accounts from the access portal, and implementing the identity account architecture that we discussed in this recipe to switch roles and log in to different AWS accounts becomes redundant. Nevertheless, roles are useful in a variety of use cases, including cross-service access, as we will see in the next recipe.

## There's more...

Let us quickly explore how we can use the **external ID** feature while switching roles for enhanced security.

The **confused deputy** problem is a security vulnerability where a program with restricted access unintentionally grants unauthorized access to its resources. This issue arises when a trusted entity delegates authority to a less-trusted one, which then acts on behalf of the trusted entity but in an unintended manner, potentially leading to security breaches. In the context of our recipe, where a source account tries to assume a role in a destination account, this vulnerability is pertinent due to the potential risk of unauthorized access to sensitive resources in the destination account by the less-trusted source account. This risk stems from the delegation of authority through role assumption mechanisms, where proper controls are necessary to prevent misuse and unauthorized access.

Now, let us see how we can solve the confused deputy problem with cross-account role switching using external IDs. An external ID serves as a unique key to establish a trust bond between two AWS accounts, commonly utilized when an AWS account intends to authorize third-party account access to its resources. Specifically, in scenarios involving cross-account IAM roles, such as the one we saw in this recipe, the resource-owning account (trusting account) designates an external ID. The third party (trusted account) must then use this external ID to assume the specified role. By requiring a matching external ID for both the role assumption and the requester, the system ensures that only authorized entities, verified through their external ID and proper permissions, can access the resources, thereby safeguarding against such vulnerabilities.

While setting up the destination account in *Figure 2.24*, the **Require external ID (Best practice when a third party will assume this role)** option was not selected. To use an external ID, we can select this option and add an external ID value of our choice. Once an external ID is configured, we need to provide the external ID while switching roles as given in the following command, otherwise, access will be denied:

```
aws sts assume-role --role-arn ROLE_ARN --role-session-name SESSION_
NAME --profile AWSSECBCUser1S --external-id awssecb-cust-0819
```

## See also

- Read more about using `external-id` at <https://www.cloudericks.com/blog/mitigating-confused-deputy-problem-external-ids-secure-cross-account-access-aws>.
- Read more about identity account architecture within AWS at <https://www.cloudericks.com/blog/implementing-identity-account-architecture-within-aws>.

## Cross-service access via IAM roles on EC2 instances

In this recipe, we will create an IAM role that allows an EC2 instance to access S3 APIs and then attach it to an EC2 instance. IAM roles provide temporary permissions for an AWS service or user to access another AWS service. This avoids the need for hard coding credentials such as access keys and secret access keys within EC2 instances.

## Getting ready

To complete the steps within this recipe, we need the following:

- A working AWS account, `awssecb-sandbox-1`, and a user with `AdministratorAccess` permission to that account, `awssecbadmin1`, following the *Technical requirements* section of this chapter.
- Working knowledge of IAM, EC2, and S3 services.

## How to do it...

We can create an IAM role for an EC2 instance with access to S3 APIs as follows:

1. Go to the **IAM** dashboard.
2. Click on **Roles** from the left sidebar.
3. Click on **Create role**.
4. Under **Trusted entity type**, select **AWS service**, and under the **Service or use case**, select **EC2** as the service that will use this role, and click **Next**.

EC2 ▼

Choose a use case for the specified service.

Use case

- ☒ **EC2**  
Allows EC2 instances to call AWS services on your behalf.
- ☐ **EC2 Role for AWS Systems Manager**  
Allows EC2 instances to call AWS services like CloudWatch and Systems Manager on your behalf.
- ☐ **EC2 Spot Fleet Role**  
Allows EC2 Spot Fleet to request and terminate Spot Instances on your behalf.
- ☐ **EC2 - Spot Fleet Auto Scaling**  
Allows Auto Scaling to access and update EC2 spot fleets on your behalf.
- ☐ **EC2 - Spot Fleet Tagging**  
Allows EC2 to launch spot instances and attach tags to the launched instances on your behalf.
- ☐ **EC2 - Spot Instances**  
Allows EC2 Spot Instances to launch and manage spot instances on your behalf.
- ☐ **EC2 - Spot Fleet**  
Allows EC2 Spot Fleet to launch and manage spot fleet instances on your behalf.
- ☐ **EC2 - Scheduled Instances**  
Allows EC2 Scheduled Instances to manage instances on your behalf.

Cancel Next

Figure 2.28 – Select trusted entity

5. Select **AmazonS3FullAccess** and click **Next**.

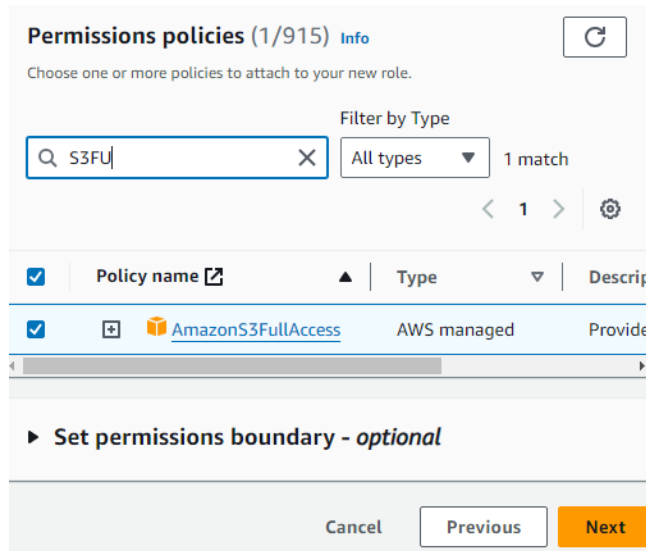


Figure 2.29 – Permissions policies

6. Give the role name (for example, `MyS3AccessRole`).
7. Optionally add tags and click on **Create Role**.

We can associate the role with an EC2 instance as follows:

- I. Go to the **EC2** dashboard.
- II. Click on **Instances** from the left sidebar.
- III. Select our instance, click on **Actions**, click on **Security**, and click on **Modify IAM role**:

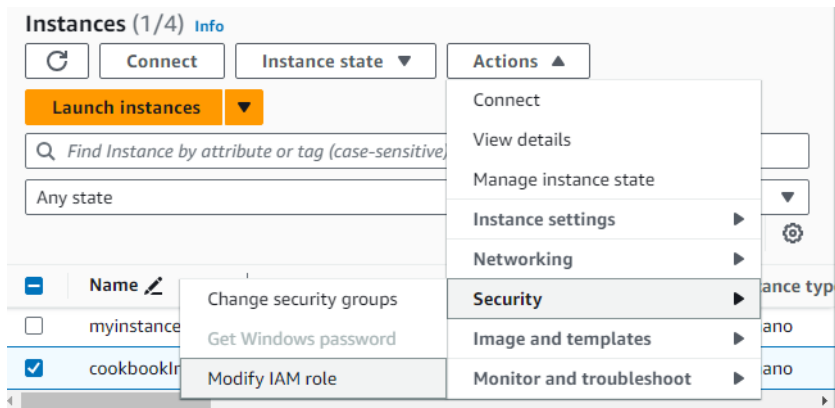


Figure 2.30 – Attaching the IAM role to an instance



For instance, IAM roles can be assigned to Lambda functions to access specific AWS resources such as S3 buckets or DynamoDB tables without requiring hard-coded credentials. Similarly, ECS tasks can assume IAM roles to interact with other AWS services during containerized application deployments. Moreover, IAM roles can be leveraged by AWS organizations to delegate permissions across multiple accounts securely.

Both IAM users and roles are IAM identities assigned specific permission policies. While users are associated with static credentials such as access keys, which pose a risk of exposure, IAM roles offer temporary security credentials for each session, reducing the reliance on long-term static credentials. Roles can be assumed by various entities including users, groups, applications, or AWS services.

In essence, IAM roles serve as a foundational element for implementing secure and efficient cross-service access controls within AWS ecosystems, promoting agility, scalability, and robust security practices.

### ***Important concepts related to IAM roles***

Let us quickly go through some more important concepts about IAM roles:

- The trust policy for a role allows a user in the trusted account to switch to or assume that role.
- A wildcard (\*) cannot be specified as a principal for a trust policy.
- When a user assumes a role, it temporarily gives up its own permissions until the user stops using the role.
- Some services allow attaching a policy directly to a resource without needing to use a role as a proxy. These resources include S3 buckets, Glacier vaults, Amazon SNS topics, and Amazon SQS queues.
- Roles can be used by external users authenticated by an external identity provider service to get access to AWS resources.
- Roles allow mobile apps to use AWS resources without embedding AWS access keys within the app.
- Role chaining is the process where a role assumes a second role through the AWS CLI or API.
- To pass the role information to an EC2 instance when the instance starts, we can add the role within an instance profile. An instance profile can be considered a container for an IAM role. The `list-instance-profiles-for-role` CLI command lists the instance profiles for a role.
- The permissions boundary is a feature we can use to set the maximum permissions that an identity-based policy can grant to an IAM entity, such as a user or role. The `put-role-permissions-boundary` CLI command can be used to create or update the permissions boundary for a role, while `delete-role-permissions-boundary` will delete the permissions boundary for the role.

- The `attach-role-policy` CLI command attaches a policy to a role, while `detach-role-policy` detaches a policy from a role.
- The `put-role-policy` CLI command creates or updates an inline policy, `get-role-policy` retrieves the specified inline policy in a role, and `delete-role-policy` deletes the specified inline policy.

## See also

We can read more about EC2 instance profiles and IAM roles at <https://www.cloudericks.com/blog/secure-cross-service-access-ec2-instance-profiles-iam-roles>.

# Key Management with KMS and CloudHSM

Encryption is like turning our message into a secret code so that only the intended recipient can understand it. Imagine that we have a message that we do not want anyone else to read. The message, in its original form, is called **plain text**. After we encode it, it becomes **cipher text**, which looks like gibberish to anyone who does not know how to decode it. Converting the plain text into cipher text is called **encryption**, and converting the cipher text back into plain text is called **decryption**. To carry out encryption and decryption, we use a set of mathematical rules known as an **encryption algorithm**. Examples of encryption algorithms include **Rivest-Shamir-Adleman (RSA)**, **Advanced Encryption Standard (AES)**, **Data Encryption Standard (DES)**, **Triple Data Encryption Standard (3DES)**, **Blowfish**, **Twofish**, **Elliptic Curve Cryptography (ECC)**, **International Data Encryption Algorithm (IDEA)**, and **Pretty Good Privacy (PGP)**.

You might be wondering, if encryption algorithms are public knowledge, what's to prevent anyone from decrypting the data? The answer lies in the use of **encryption keys** – unique strings of characters that safeguard data when used alongside the algorithm. There are two primary types of encryptions: **symmetric** and **asymmetric**. Symmetric encryption employs the same key for both encryption and decryption, necessitating that this key be kept confidential between the parties involved. Asymmetric encryption, in contrast, utilizes a key pair – one key for encryption and the other for decryption. For instance, a public key, which can be openly shared, can be used for encryption, and a private key, which is strictly guarded by the intended recipient, can be used for decryption. Thus, the privacy of encrypted data remains protected, even with the widespread knowledge of the algorithm, due to the confidential nature of the encryption keys.



AWS **Key Management Service (KMS)** is the primary service within the AWS cloud that helps us create and manage encryption keys and is the primary focus of this chapter. KMS supports both symmetric and asymmetric encryption keys. Based on ownership, we can categorize KMS keys as **customer-managed keys**, **AWS-managed keys**, and **AWS-owned keys**. With customer-managed keys, we create and manage keys. With AWS-managed keys, AWS creates a key in our account for a particular service, such as S3, EBS, and so on, and manages the key for us. AWS also has another key type called AWS-owned keys that AWS creates and manages for use cases such as default encryption for services such as S3. With both customer-managed and AWS-managed keys, we have visibility into various keys and their usage through CloudTrail logs. However, with AWS-owned keys, we do not have any visibility.

In this chapter, we will learn how to work with AWS **CloudHSM**. CloudHSM is another service within AWS that allows us to manage encryption keys but uses dedicated HSMs for enhanced security. KMS makes use of shared **hardware security modules (HSMs)**.

In this chapter, we will cover the following recipes:

- Creating keys in KMS
- Creating keys with external key material (BYOK)
- Rotating keys in KMS
- Granting permissions programmatically with grants
- Using key policies with conditional keys
- Sharing customer-managed keys across accounts
- Creating, initializing, and activating a CloudHSM cluster

## Technical requirements

Before diving into the recipes in this chapter, we need to ensure we have the following requirements in place:

- We need an active AWS account to complete the recipes in this chapter. We can use an account that is part of AWS Organizations or a standalone account. I will be using the `awssecb-sandbox-1` account that we created in the *Multi-account management with AWS Organizations* recipe in *Chapter 1*. However, I won't be utilizing any AWS Organizations features, meaning you can follow these steps with a standalone account too.
- For administrative actions, we need a user who has **AdministratorAccess** permission to the AWS account we are working with. This can be an **IAM Identity Center user** or an IAM user. I will be using the IAM Identity Center user `awssecbadmin1` we created in the *User management and SSO with IAM Identity Center* recipe in *Chapter 1*. However, I won't be utilizing any IAM Identity Center features, meaning you can follow these steps with an IAM user, too, if the user has **AdministratorAccess** permission within the account. You can create an IAM user by following the *Setting up IAM, account aliases, and billing alerts* recipe in *Chapter 1*.

- Basic knowledge of encryption, including symmetric keys, asymmetric keys, and **public key infrastructure (PKI)** will help you understand the recipes within this chapter.

The code files for this book are available at <https://github.com/PacktPublishing/AWS-Security-Cookbook-Second-Edition>. The code files for this chapter are available at <https://github.com/PacktPublishing/AWS-Security-Cookbook-Second-Edition/tree/main/Chapter03>.

## Creating keys in KMS

In this recipe, we will create a customer-managed KMS key with the key type set to **symmetric key**. A symmetric key is the most common key that we will create with KMS. It is also worth noting that KMS keys, which are the primary resources within KMS, were once known as **customer master keys (CMKs)**. This renaming helps avoid confusion with the term customer-managed keys, which could also be abbreviated as CMKs.

### Getting ready

We'll need the following to complete this recipe:

- A working AWS account (`awssecb-sandbox-1`) and a user (`awssecbadmin1`), as described in the *Technical requirements* section.
- Two users or roles. These could be IAM users or roles, including those corresponding to IAM Identity Center users. I will be using the `awssecb_admin1` user as the **key administrator**. Key administrators can administer the key through the KMS API. I will use another user, `awssecb_user1`, as the **key user**. Key users can use the customer-managed key to encrypt and decrypt data. We can even use the same user as both a key administrator and a key user. I will also explain how we can use a role corresponding to an IAM Identity Center user as a key administrator or key user.

### How to do it...

We can create a customer-managed key in KMS from the console as follows:

1. Log in to the AWS Management Console and go to **Key Management Service**.
2. Click on **Customer-managed keys** from the left sidebar.
3. Click **Create key**.

4. In the **Configure key** step, as shown in the following figure, select **Symmetric** for **Key type**, and choose **Encrypt and decrypt** for **Key usage**:

**Configure key**

**Key type** [Help me choose](#)

☒ **Symmetric**  
A single key used for encrypting and decrypting data or generating and verifying HMAC codes.

☐ **Asymmetric**  
A public and private key pair used for encrypting and decrypting data or signing and verifying messages.

**Key usage** [Help me choose](#)

☒ **Encrypt and decrypt**  
Use the key only to encrypt and decrypt data.

☐ **Generate and verify MAC**  
Use the key only to generate and verify hash-based message authentication codes (HMAC).

► **Advanced options**

**Cancel** **Next**

Figure 3.1 – Configuring Key type and Key usage

5. Expand **Advanced options** and verify that **Key material origin** is set to **KMS - recommended** and **Regionality** is set to **Single-region key**, as shown in the following figure. Then, click **Next** to proceed to the **Add labels** steps:

▼
Advanced options

Key material origin

Key material origin is a KMS key property that represents the source of the key material when creating the KMS key. [Help me choose](#)

☒
**KMS - recommended**

AWS KMS creates and manages the key material for the KMS key.

☐
**External (Import Key material)**

You create and import the key material for the KMS key.

☐
**AWS CloudHSM key store**

AWS KMS creates the key material in the AWS CloudHSM cluster of your AWS CloudHSM key store.

☐
**External key store**

The key material for the KMS key is in an external key manager outside of AWS.

Regionality

Create your KMS key in a single AWS Region (default) or create a KMS key that you can replicate into multiple AWS Regions. [Help me choose](#)

☒
**Single-region key**

Never allow this key to be replicated into other regions

☐
**Multi-region key**

Allow this key to be replicated into other regions

Cancel

Next

Figure 3.2 – Advanced options

- In the **Add labels** step, enter `seccb-dev-encryption` for **Alias** and `Dev Encryption Key for the Sec CB project` for **Description**. Optionally, add tags by clicking on the **Add tag** button. Click **Next** to proceed to the **Define key administrative permissions** step.
- In the **Define key administrative permissions** step, select **Key administrators**, who are the IAM users and roles who can administer this key through the KMS API. I have selected `awsseccb_admin1`. However, as we can see, if we are using an IAM Identity Center user, we can select a role corresponding to the combination of that IAM Identity Center user and a **permission set**.

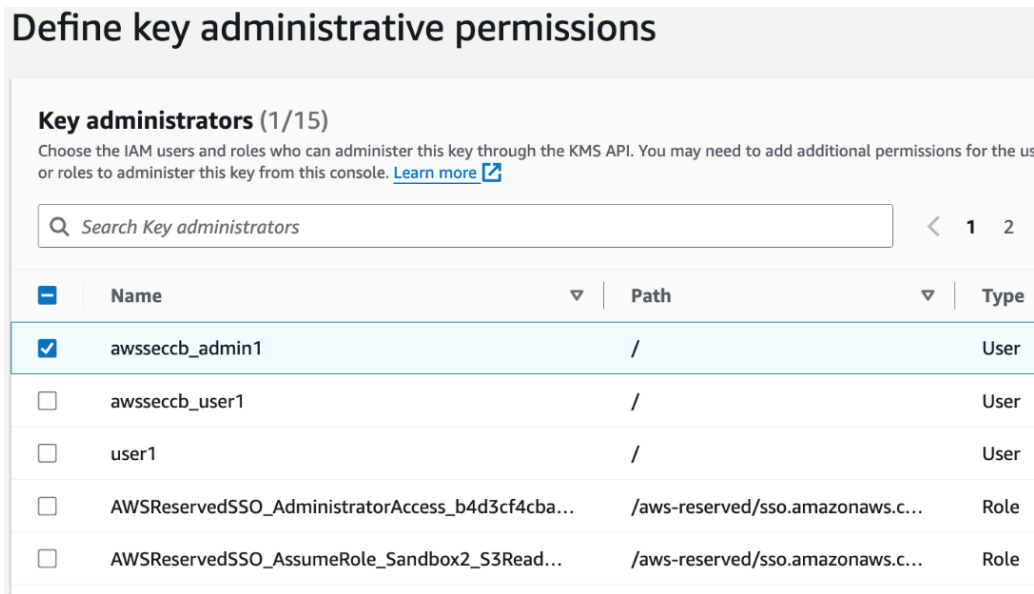


Figure 3.3 – Defining key administrative permissions

- 8. For **Key deletion**, check **Allow key administrators to delete this key**. Click **Next** to proceed to the **Define key usage permissions** step.
- 9. Regarding the **Define key usage permissions** step, select **Key users**, who are IAM users and roles who can use the customer-managed key to encrypt and decrypt data. In the same step, we can optionally add **Other AWS accounts** that can use this key. I have selected the `awsseccb_user1` user. Similar to *Step 7*, if we are using an IAM Identity Center user, we can select a role corresponding to the combination of that IAM Identity Center user and a permission set. Click **Next**.
- 10. Review **Key configuration**, **Alias and description**, **Tags**, and **Key policy** in JSON format. Click **Finish** to create the key. We should see a message stating that the key has been created, as well as details of the key ID, as shown in the following figure:



Figure 3.4 – Key creation success message

The key policy that's generated by AWS is kept within the code files for reference as `generated-key-policy.json`.

In this recipe, we created a KMS key. We will go through some important concepts related to KMS keys in the next section.

## How it works...

Creating encryption keys within AWS KMS involves a straightforward process. First, we need an active AWS account with an IAM or IAM Identity Center user with administrative privileges to create and manage keys. We should have the `kms:CreateKey` permission to create new KMS keys. To add tags, we need `kms:TagResource` permission. We used a user with **AdministratorAccess** for this recipe and hence had both of these permissions.

While configuring the key, we selected the symmetric KMS key option for general encryption and decryption purposes, as shown in *Figure 3.1*. In the **Advanced options** section, we selected the **key material** origin as KMS, which is the recommended option. In the world of cryptography, key material refers to the string of bits that are used in a cryptographic algorithm and is the core element that encrypts and decrypts our data. Each AWS KMS key is associated with a key material, as referenced in its metadata. By default, AWS is responsible for creating the key material using FIPS-validated hardware security modules, and it never leaves AWS KMS unencrypted. The sole exception lies in the public keys of asymmetric key pairs, which can be exported for outside use.

The key material's origin is a specific attribute of a KMS key in AWS KMS, which indicates the source from which its key material is derived. The key material origin values for a KMS key can be one of the following, as we saw in *Figure 3.2*:

- **KMS - recommended** is the default and most recommended. If we select this option, AWS handles the creation and management of the key material.
- **External (Import key material)** is for people who prefer to import their own key material. This requires us to manage and secure this material.
- With **AWS CloudHSM key store**, AWS KMS creates the key material in our AWS CloudHSM cluster.
- **External key store** is used when the key material is in an external key manager outside AWS. This is specific to KMS keys in an external key store.

As we saw in *Figure 3.3*, we can assign IAM users or roles as key administrators. Key administrators are the IAM users and roles who can administer this key through the KMS API. We also chose the option to allow key administrators to delete the key. Similarly, we can assign IAM users or roles as key users. Key users are IAM users and roles that can use the customer-managed key to encrypt and decrypt data.

A KMS key policy is a JSON document that specifies who can access the key and under what conditions. Along with the key, AWS will create a default key policy and assign required permissions for the key administrators and key users we select. If we explore the generated key policy, we should be able to see the `kms:Create*`, `kms:Describe*`, `kms:Enable*`, `kms:List*`, `kms:Put*`, `kms:Update*`, `kms:Revoke*`, `kms:Disable*`, `kms:Get*`, `kms>Delete*`, `kms:TagResource`, `kms:UntagResource`, `kms:ScheduleKeyDeletion`, and `kms:CancelKeyDeletion` permissions for key administrators. For key users, we should see the `kms:Encrypt`, `kms:Decrypt`, `kms:ReEncrypt*`, `kms:GenerateDataKey*`, and `kms:DescribeKey` permissions. These permissions should also give us an idea about the operations we can perform within KMS.

## There's more...

In this recipe, we created a key using the standard options available within KMS. We can configure these options based on our requirements. For example, in this recipe, we chose the option to allow key administrators to delete the key. Based on our security requirements, we may unselect this option to not allow key administrators to delete the key.

### Tip

Play around with all the options available within KMS and get familiar with them. This will help you decide on what options to choose when you are faced with a scenario at work, in an exam, or in an interview.

Let's quickly go through some important points about the AWS KMS service:

- We created a symmetric key in this recipe. AWS KMS also has support for asymmetric keys. AWS KMS allows you to create and manage asymmetric key pairs for encryption and signing, facilitating scenarios that require PKI, such as encrypting data so that only the private key holder can decrypt it, or signing data for sender verification.
- AWS KMS keys can encrypt data up to 4 KB in size and hence are generally used to encrypt other keys, such as **data keys**. These data keys are then used to encrypt the actual data. These data keys are not created and managed within the AWS KMS service.
- KMS is a region-specific service and, hence, the keys managed by KMS are region-specific. Therefore, to use KMS keys, the respective services should also be in the same region. For example, to use the key that we created for encrypting S3 data, the S3 bucket needs to be in the same region.
- KMS supports a feature called **multi-region keys**, which allows data to be decrypted in a region different from the one where it was encrypted. It is important to note that these keys are not currently global. Instead, there is a primary key that is replicated to replica keys in other regions.
- A user with S3 administrator permissions will not have permission to view a file encrypted using KMS key encryption unless they are a key user for the key being used to encrypt that file.
- Key administrators do not have permission to encrypt or decrypt data using those keys. Key administrators, however, can modify the key policy to add themselves as key users. This is where the audit and logging services become important.
- AWS KMS integrates with AWS CloudTrail to provide logs of all key usage to help meet compliance and regulatory requirements. This integration allows users to track when and who used KMS keys for encryption and decryption, providing an audit trail that can be crucial for forensic analysis and compliance audits.

- We cannot directly delete a key. A key administrator can disable and/or schedule a key for deletion. At the time of writing, the administrator can specify a waiting period between 7 and 30 days inclusive while scheduling a key deletion. With multi-region keys, we need to first delete the replica keys; only then can we delete the primary key. Therefore, the minimum number of days to delete a multi-region key is 14 days: 7 days for deleting the replica keys and 7 days for deleting the primary key.
- Once a key has been disabled, we cannot decrypt any data that is encrypted with that key until we enable that key again.
- Once a key has been deleted, we cannot decrypt any data encrypted with that key.

## See also

- A good understanding of encryption can help you better understand the AWS KMS service. You can learn about encryption here: <https://www.secdops.com/blog/getting-started-with-encryption>.
- AWS KMS is a service with a lot of features. You can start reading more about KMS and its features here: <https://www.cloudericks.com/blog/getting-started-with-aws-key-management-service-kms>.

## Creating keys with external key material (BYOK)

When we create keys within AWS KMS, AWS creates and manages the key material for that key. We can also create keys using our own key material that has been created outside of AWS. In this recipe, we will learn how to import a key material into AWS KMS. Using external key material for our keys is called **Bring Your Own Key (BYOK)** and is useful for organizations that have strict compliance or policy requirements that mandate the use of keys they control. This key should be a 256-bit symmetric key. Asymmetric keys are not supported for BYOK.

## Getting ready

We'll need the following to complete this recipe:

- A working AWS account, `awssecb-sandbox-1`, and a user, `awssecbadmin1`, as described in the *Technical requirements* section.
- The latest OpenSSL setup on our local machine. If it's not already installed, go to the OpenSSL website at <https://www.openssl.org>, download the latest version of OpenSSL, and set it up locally while following the instructions. Run the `openssl version` command to make sure OpenSSL is installed with the latest version.



**Important note**

If you're a macOS user, you might encounter an important compatibility issue: macOS defaults to using LibreSSL with the `openssl` command, not OpenSSL. To ensure you're using the correct version of OpenSSL, you may need to install it via Homebrew and invoke it directly while specifying the full path or even adjust your system's `PATH` settings. A simple step-by-step guide on how to do this is provided in the following blog post: <https://www.secdops.com/blog/using-openssl-alongside-the-default-libressl-on-macos>.

Assuming we have set up and can run the `openssl version` command correctly, let's proceed to use it to create external key material.

**How to do it...**

We will start by creating our key from AWS KMS by setting the key material origin to **External**. Then, we will download the key import wrapper from the AWS KMS service, generate the key on our local machine, and wrap it with the import wrapper. Finally, we will upload our wrapped key material with the import wrapper to finish key creation.

***Creating a key configuration for an external key***

We can create a key configuration for an external key as follows:

1. Log in to the AWS Management Console and go to **Key Management Service**.
2. Click on **Customer-managed keys** from the left sidebar and click **Create key** at the top right of the page.
3. In the **Configure key** step, for **Key type**, select **Symmetric**, and for **Key usage**, select **Encrypt and decrypt**.
4. Expand the **Advanced options** menu and select **External (Import Key material)**. Then, check the checkbox to agree that you understand the security and durability implications while using an external key. For **Regionality**, select **Single-region key**. Click **Next** to proceed to the **Add labels** step.
5. In the **Add labels** step, set **Alias** to `seccb-dev-external-key` and **Description** to **External Key for the Dev env**. Optionally, add tags by clicking on the **Add tag** button. Click **Next** to proceed to the **Define key administrative permissions** step.
6. In the **Define key administrative permissions** step, select **Key administrators**, who are the IAM users and roles who can administer this key through the KMS API. I have selected the `awsseccb_admin1` user, as shown in *Figure 3.3*. If you're using an IAM Identity Center user, you can select a role corresponding to the combination of that IAM Identity Center user and a permission set.
7. For **Key deletion**, check **Allow key administrators to delete this key**. Click **Next** to proceed to the **Define key usage permissions** step.

8. Regarding the **Define key usage permissions** step, select **Key users** – that is, IAM users and roles who can use the customer-managed key to encrypt and decrypt data. In the same step, optionally, you can add **Other AWS accounts** that can use this key. I have selected the `awssecbcb_user1` user. If you're using an IAM Identity Center user, you can select a role corresponding to the combination of that IAM Identity Center user and a permission set. Click **Next**.
9. Review **Key configuration**, **Alias and description**, **Tags**, and **Key policy** in JSON format. Click **Finish** to create the key. The key will be created, as shown in *Figure 3.4*. You should also see the **Download wrapping public key and import token** step now.
10. Regarding the **Download wrapping public key and import token** step, under the **Configuration** section, for **Select wrapping key spec**, select **RSA\_4096 - recommended**, and for **Select wrapping algorithm**, select **RSAES\_OAEP\_SHA\_256**.

## Download wrapping public key and import token

To import key material, first select the wrapping key spec and wrapping algorithm you will use to encrypt the key material, then download the wrapping public key and import token for this AWS KMS key. [Learn more](#)

### Configuration

#### Select wrapping key spec

The key spec of the wrapping public key determines the length of the keys in the RSA key pair that protects your key material during its transport to AWS KMS.

☒ RSA\_4096 - recommended

☐ RSA\_3072

☐ RSA\_2048

#### Select wrapping algorithm

Choose the encryption algorithm that you'll use to protect ('wrap') your key material in transit to AWS KMS.

RSAES\_OAEP\_SHA\_256

### Download

Wrapping public key


WrappingPublicKey.bin

Import token

ImportToken.bin

Read me instructions

README.txt

 This wrapping public key and import token will expire in 24 hours.

 Download wrapping public key and import token

Cancel

Next

Figure 3.5 – Download wrapping public key and import token

11. Click **Download wrapping public key and import token** to download the token on your desktop. Then, click **Next** to reach the **Upload your wrapped key material** page. Keep the downloaded import parameters file for the steps we will follow shortly.

In this section we created a key configuration. In the next section, we will generate our key material and then return to this screen and click **Next** to go to the **Upload your wrapped key material** step. If we check the **Customer managed keys** page, we should see that **Status** is now set to **Pending import**.

### *Generating key material using OpenSSL*

Assuming we have set up OpenSSL as mentioned in the *Getting ready* section, we can follow these steps to generate key material and encrypt it using the wrapping key provided by AWS KMS in the previous section:

1. As we are using a symmetric key, we can use the following command to generate 256 bytes of random data and save it to a file named `MyExternalKeyMaterial.bin`. This will serve as our key material:

```
openssl rand -out ExternalKeyMaterialPlaintext.bin 32
```

This will generate a file called `ExternalKeyMaterialPlaintext.bin`.

2. Execute the following command from the same folder while providing the name of our wrapping key for the `inkey` parameter:

```
openssl pkeyutl -encrypt -in ExternalKeyMaterialPlaintext.bin
-out ExternalKeyMaterialEncrypted.bin -inkey WrappingPublicKey.
bin -keyform DER -pubin -pkeyopt rsa_padding_mode:oaep -pkeyopt
rsa_oaep_md:sha256 -pkeyopt rsa_mgf1_md:sha256
```

This will generate a file called `ExternalKeyMaterialEncrypted.bin`.

With that, we've generated the key material. In the next section, we will upload this key material to AWS.

### *Continuing with key creation from the Management Console*

We can upload our key material in the AWS Management Console as follows:

1. Go back to the **Upload your wrapped key material** step's page in the console. This is where we stopped in the *Creating a key configuration for an external key* section.
2. If we are not on the **Upload your wrapped key material** step's page, we can go to **Key Management Service** via the dashboard, click **Customer-managed keys** from the left sidebar, and then click on the hyperlink under **Aliases** for the key for which we downloaded the wrapping key. Go to the **Key material** tab and click **Import key material**. Then, click **Next** to go to the **Upload your wrapped key material** step's page.
3. On the **Upload your wrapped key material** step's page, click **Choose file** under **Wrapped key material** and select the `ExternalKeyMaterialEncrypted.bin` file.

4. Click **Choose file** under **Import token** and select the import token we downloaded from KMS in the previous section.
5. Leave the **Key material expires - optional** option unchecked and click **Upload key material**. We should see a message that the key has been uploaded.

The key is now ready for use. If we check the **Customer managed keys** page, we should see that **Status** has changed from **Pending import** to **Enabled**.

## How it works...

In this recipe, we created an AWS KMS key by setting the key material origin to **External**. After that, we selected one of the allowed encryption schemes for the key wrapper and downloaded the key wrapper. This key wrapper is the public key that is used to encrypt and securely upload our key material to the AWS KMS service. An import token from the AWS KMS service was also downloaded, along with the key wrapper. The import token is used to make sure the key that's uploaded is the right one for the key that we downloaded for the wrapper token.

## There's more...

In this recipe, we used **SHA\_256** to wrap our external key material before uploading it. We may also use **SHA\_1**, but it is less secure. If we are using SHA-1, we can generate the encrypted external key material, `ExternalKeyMaterialEncrypted.bin`, with the following command:

```
openssl pkeyutl -encrypt -in ExternalKeyMaterialPlaintext.bin -out  
ExternalKeyMaterialEncrypted.bin -inkey WrappingPublicKey.bin -keyform  
DER -pubin -pkeyopt rsa_padding_mode:oaep -pkeyopt rsa_oaep_md:sha1
```

Let's quickly go through some more details regarding importing keys into AWS KMS:

- When we import our key material, we are responsible for generating the key material with randomness, as per our security requirements. We are also responsible for the durability of the key material.
- With imported key material, we can set an expiration date for the key material and also manually delete it. We can make the key available again in the future by importing the key material into the KMS key.
- We cannot delete the key material for a KMS key with AWS key material. We can, however, schedule the deletion of that KMS key with 7-30 days' notice.
- Once a KMS key has been deleted, any data that's been encrypted by it cannot be decrypted. This is true for both KMS keys with AWS key material and KMS keys with imported key material.
- A key material imported into a KMS key is permanently associated with that KMS key.
- We can reimport the key material. However, we cannot import a different key material again for that KMS key.

- A ciphertext encrypted with a KMS key with an external key material cannot be decrypted by another KMS key, even if we use the same key material.
- A KMS key with an imported key material must be deleted before we can reimport the key material again into another KMS Key.
- We can reimport the key material into an existing KMS key if the key material is deleted.
- In the case of region-wide failures that affect KMS keys, AWS won't automatically restore any imported key material. In such scenarios, we need to have a copy of our key material to reimport it.

## See also

You can read more about external keys and BYOK in the following blog post: <https://www.cloudericks.com/blog/aws-kms-with-external-key-material-the-byok-solution>.

## Rotating keys in KMS

**Key rotation** refers to the process of changing the encryption key that's used to secure our data. This practice is crucial for minimizing risk in case a key is compromised. AWS supports automatic and manual key rotation for customer-managed keys.

Rotating keys regularly is a best practice that needs to be followed while using keys. Key rotation may also be a requirement based on regulatory rules or corporate policies. These rules and policies may also provide guidelines on the key rotation frequencies. We will look at the different cases of key rotation in this recipe.

## Getting ready

We'll need the following to complete this recipe:

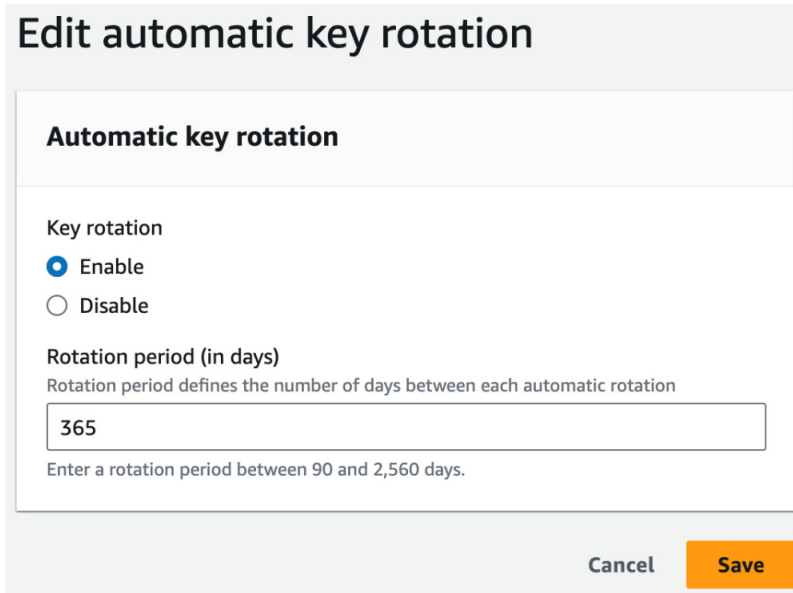
- A working AWS account, `awssecb-sandbox-1`, and a user, `awssecbadmin1`, as described in the *Technical requirements* section.
- A customer-managed KMS key with AWS key material. I will be using the key with the `secb-dev-encryption` alias that we created in the *Creating keys in KMS* recipe.

## How to do it...

We can specify automatic key rotation every year (365 days) for customer-managed KMS keys with AWS key material as follows:

1. Log in to the AWS Management Console and go to **Key Management Service**.
2. Click on **Customer-managed keys** on the navigation pane to list all the keys we have created.

3. Click on the **Aliases** or **Key ID** property of our customer-managed key for which we need to do the rotation.
4. Go to the **Key rotation** tab and click **Edit**.
5. For **Key rotation**, select **Enable** and set **Rotation period (in days)** with a value of 365, as shown in the following figure:



**Edit automatic key rotation**

**Automatic key rotation**

Key rotation

☒ Enable

☐ Disable

Rotation period (in days)

Rotation period defines the number of days between each automatic rotation

365

Enter a rotation period between 90 and 2,560 days.

Cancel Save

Figure 3.6 – Automatic key rotation

6. Click **Save**. We will get a notification that the operation was successful.

With that, we've learned how to enable automatic key rotation. We will learn more about key rotation in the subsequent sections of this recipe.

## How it works...

AWS-managed keys are automatically rotated every year. Previously, it was every 3 years. For customer-managed keys, AWS supports both automatic and manual key rotation. With automatic rotation, only the backing key of a KMS key is rotated. This means that the KMS key ID, ARN, region, policies, permissions, and other properties remain the same. Therefore, we do not need to change applications or aliases that use this KMS key.

In this recipe, we selected the option to automatically rotate our KMS key every year. AWS will now rotate the key every year but keep a copy of the old backing key to decrypt any data that was encrypted with the old backing key. AWS keeps the older backing keys until we delete them.

## There's more...

Let's quickly go through some of the important points related to AWS KMS key rotation:

- Automatic key rotation every year (365 days) is only supported for KMS keys with AWS key materials.
- We can do a manual key rotation for KMS keys with AWS key material if we want a different duration for the key rotation.
- With automatic key rotation, only the KMS key is rotated, not the data keys encrypted with it.
- With automatic key rotation, new encryptions are made using the new backing key. However, data encrypted using an older backing key is decrypted using that old key. For this purpose, AWS keeps all backing keys available until we delete the KMS key.
- With automatic key rotation, even if we disable key rotation, old backing keys will still be available to decrypt data that was encrypted using that key.
- With automatic key rotation, if you disable rotation and re-enable it again, it will continue with the old key rotation schedule if the backing key is less than a year old. If the backing key is older than 365 days, it is rotated immediately and then rotated again every 365 days.
- With automatic key rotation, key rotation will not happen if a key is pending deletion. If deletion is canceled, it will continue with the old key rotation schedule if the backing key is less than a year old. If the backing key is older than 365 days, it is rotated immediately and then rotated again every 365 days.
- Automatic key rotation is not supported for custom key stores backed by AWS CloudHSM clusters. For such KMS keys, the value of the **Origin** field is `AWS_CloudHSM`. In this case, we need to manually rotate keys and change any encrypted data or aliases to use the new key.
- For AWS-managed keys, we cannot change the rotation frequency, which is currently 1 year.
- Automatic key rotation can be monitored using the KMS key rotation event in EventBridge.
- We can use the AWS KMS API to enable and disable automatic key rotation. It is good practice to use aliases to refer to KMS keys when we do manual key rotation. We can update the alias so that it points to the new target KMS key instead of the old one.
- Even with manual rotation, AWS KMS can identify the right backing key that was used for encryption and use it for decryption, so long as we keep the older KMS keys available.
- We can update aliases using the `update-alias` subcommand of the AWS KMS API.

## See also

You can read more about AWS KMS key rotation, including manual key rotation, at <https://www.cloudericks.com/blog/understanding-aws-kms-key-rotation>.

## Granting permissions programmatically with grants

**KMS grants** can be used to give temporary granular permissions to AWS KMS API operations such as encrypting, decrypting, and describing keys, and more. We can use grants to provide access to a user in their account or even another account. In this recipe, we will grant access to a user so that they can encrypt and decrypt files using AWS KMS.

### Getting ready

We'll need the following to complete this recipe:

- A working AWS account with two users: a user with **AdministratorAccess** permission and a user with no permissions. The CLI profiles should be configured for these users. I will be calling these users and their CLI profiles `Adminuserprofile` and `testusernopermission`, respectively, following the recipes in *Chapter 1*.
- A KMS key. We can create one by following previous recipes in this chapter. Alternatively, use the following command to create a KMS key from the AWS CLI:

```
aws kms create-key --profile Adminuserprofile
```

This will provide an output similar to the following:

```
{
  "KeyMetadata": {
    "AWSAccountId": "201882936474",
    "KeyId": "ea7136c3-7d8a-4ed4-89cc-4ca0af7d6958",
    "Arn": "arn:aws:kms:us-east-1:201882936474:key/ea7136c3-7d8a-4ed4-89cc-4ca0af7d6958",
    "CreationDate": "2023-12-13T18:01:24.897000+05:30",
    "Enabled": true,
    "Description": "",
    "KeyUsage": "ENCRYPT_DECRYPT",
    "KeyState": "Enabled",
    "Origin": "AWS_KMS",
    "KeyManager": "CUSTOMER",
    "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
    "KeySpec": "SYMMETRIC_DEFAULT",
    "EncryptionAlgorithms": [
      "SYMMETRIC_DEFAULT"
    ],
    "MultiRegion": false
  }
}
```

Figure 3.7 – Creating a key using the CLI

Before proceeding, we have to check whether our test user has any permissions by running the following command. Ensure you replace my `key-id` with your `key-id`:

```
aws kms encrypt --plaintext "$(echo -n 'hello heartin' |
base64)" --key-id ea7136c3-7d8a-4ed4-89cc-4ca0af7d6958 --profile
testusernopermission
```



We should get an error message similar to the one shown in the following screenshot:

```
An error occurred (AccessDeniedException) when calling the Encrypt operation
: User: arn:aws:iam::201882936474:user/testuser is not authorized to perform
: kms:Encrypt on resource: arn:aws:kms:us-east-1:201882936474:key/ea7136c3-7
d8a-4ed4-89cc-4ca0af7d6958 because no identity-based policy allows the kms:E
ncrypt action
```

Figure 3.8 – Response to the user permission check

In the preceding command, we can specify the key ID alone (as we did), the complete key ARN, or an alias (if one is available).

Next, we will learn how to use grants so that we can give permissions programmatically.

## How to do it...

We can grant encrypt permission to `testuser` and then use it to encrypt as follows:

1. We can get the user's ARN from the IAM dashboard or prepare one based on the preceding format. We can also use the `aws iam get-user` command to get the user's ARN from the console:

```
aws iam get-user --user-name testuser --profile Adminuserprofile
```

This command will return a response similar to the following:

```
{
  "User": {
    "Path": "/",
    "UserName": "testuser",
    "UserId": "AIDAS6AJJESNIYK4QGADS",
    "Arn": "arn:aws:iam::201882936474:user/testuser",
    "CreateDate": "2023-12-13T12:19:25+00:00",
    "Tags": [
      {
        "Key": "AKIAS6AJJESNESOTS6E5",
        "Value": "wertyuioip"
      }
    ]
  }
}
```

Figure 3.9 – Response for the get-user command

2. Grant `Encrypt` permission to `testuser` using the `create-grant` subcommand by providing the user's ARN:

```
aws kms create-grant --key-id ea7136c3-7d8a-4ed4-89cc-
4ca0af7d6958 --grantee-principal arn:aws:iam::201882936474:user/
testuser --operations "Encrypt" --profile Adminuserprofile
```

We should get a response similar to the one shown in the following screenshot:

```
{
  "GrantToken": "AQpAZTZmYjE0NTAzZTA4MzE1ZmY0NGJjYWE2MTM0M2RjY2Z2hNjAyYjc1OGRkZGQ0MjhMTdmZWQyMTdkNjBiODQyNyKIAGEBAGB45vsUUD4IMV_0S8qmE0Pcz6Yct1jd3UKKF_7SF9YLhCcAADfMIHcBgkqhkiG9w0BBwaggc4wgcsCAQAwgcUGCSqGSIB3DQEHATAeBgIghkgBZQMEAS4wEQQMfifb70be7g9bt7QdcAgEQgIGXDQrsyIBb-phD215m-470WGpyYAnK-scBG2a609FrGZ51r_vGut7L9ERCZmAPQWboucJ3UqoA2IfavBgD-F3PNdqtWdZEmkFfbTrbXkp0MqqzjxSoSdJW2f0GSGOHjCUcsZbI4LH6uHebZrPQx-SatSQbZW9senYHNMBUwjEQUhTQ0U4i22Kdn01xyRoH8h_cxEXNgeua5Sog5lefz-yNhM6aSDmKXjNGbeR7deBvmbTh7dF4kUxVm7U",
  "GrantId": "e6579fcfec8d84ce9a48398a5e33466de47b75e06f99b4e1edd178914c559bb5"
}
```

Figure 3.10 – Response for the create-grant subcommand

3. Encrypt data with `testuser` using the `encrypt` subcommand:

```
aws kms encrypt --plaintext "$(echo -n 'hello heartin' |
base64)" --key-id ea7136c3-7d8a-4ed4-89cc-4ca0af7d6958 --profile
testusernopermission
```

This time, we should get a successful response, as follows:

```
{
  "CiphertextBlob": "AQICAHjL9XXmYrZJqDidwxAtdUtUsk1h71Kegs8+sgzkT18hNwEgtZuqFCEBk0Ewqbdw0EnVAAAABTBrBgkqhkiG9w0BBwagXjBcAgEAMFcGCSqGSIB3DQEHATAeBgIghkgBZQMEAS4wEQQMfifb70be7g9bt7QdcAgEQgIGXDQrsyIBb-phD215m-470WGpyYAnK-scBG2a609FrGZ51r_vGut7L9ERCZmAPQWboucJ3UqoA2IfavBgD-F3PNdqtWdZEmkFfbTrbXkp0MqqzjxSoSdJW2f0GSGOHjCUcsZbI4LH6uHebZrPQx-SatSQbZW9senYHNMBUwjEQUhTQ0U4i22Kdn01xyRoH8h_cxEXNgeua5Sog5lefz-yNhM6aSDmKXjNGbeR7deBvmbTh7dF4kUxVm7U",
  "KeyId": "arn:aws:kms:us-east-1:201882936474:key/ea7136c3-7d8a-4ed4-89cc-4ca0af7d6958",
  "EncryptionAlgorithm": "SYMMETRIC_DEFAULT"
}
```

Figure 3.11 – Response for the encrypt command after getting the permission

4. Verify the grants for the key using the `list-grants` subcommand:

```
aws kms list-grants --key-id ea7136c3-7d8a-4ed4-89cc-4ca0af7d6958 --profile Adminuserprofile
```

This should return a response similar to the following:

```
{
  "Grants": [
    {
      "KeyId": "arn:aws:kms:us-east-1:201882936474:key/ea7136c3-7d8a-4ed4-89cc-4ca0af7d6958",
      "GrantId": "e6579fcfec8d84ce9a48398a5e33466de47b75e06f99b4e1edd178914c559bb5",
      "Name": "",
      "CreationDate": "2023-12-13T18:24:29+05:30",
      "GranteePrincipal": "arn:aws:iam::201882936474:user/testuser",
      "IssuingAccount": "arn:aws:iam::201882936474:root",
      "Operations": [
        "Encrypt"
      ]
    }
  ]
}
```

Figure 3.12 – Response for the list-grant command

5. Revoke the grant with the `revoke-grant` subcommand:

```
aws kms revoke-grant --key-id ea7136c3-7d8a-4ed4-89cc-4ca0af7d6958 --grant-id e6579fcfec8d84ce9a48398a5e33466de47b75e06f99b4e1edd178914c559bb5 --profile Adminuserprofile
```

6. We can verify that the grant has been revoked by trying to encrypt it using `testuser` and by running the `list-grants` subcommand. Run the `encrypt` command, similar to *Step 2* of this recipe. We should now get an error message similar to the following:

```
An error occurred (AccessDeniedException) when calling the Encrypt operation : User: arn:aws:iam::201882936474:user/testuser is not authorized to perform kms:Encrypt on resource: arn:aws:kms:us-east-1:201882936474:key/ea7136c3-7d8a-4ed4-89cc-4ca0af7d6958 because no identity-based policy allows the kms:Encrypt action
```

Figure 3.13 – Response for the `encrypt` command after revoking the permission

7. Run the `list-grant` subcommand, similar to *Step 3* of this recipe. We should now get a response similar to the following:

```
{
  "Grants": []
}
```

Figure 3.14 – Response for the `list-grant` command after revoking all the grants

Similarly, we can grant permission for other operations.

## How it works...

In this recipe, we granted permission to a user using the `create-grant` subcommand of the `aws kms` CLI command. We verified that the user could not perform encryption before granting permission. We verified that the user could perform encryption after granting permission.

Then, we revoked the grant using the `revoke-grant` subcommand of the `aws kms` CLI command. We also used other subcommands, such as `list-grants` to list the grants for a particular key ID and `encrypt` to encrypt the plain text.

## There's more...

In this recipe, we granted permission to only one operation. We can grant permission to multiple operations, as shown here:

```
aws kms create-grant --key-id 1ab77c7a-7ca4-4387-a4c5-2fba3cb5c0f5 --grantee-principal arn:aws:iam::135301570106:user/testuser --operations "Encrypt" "Decrypt" --profile Adminuserprofile
```

---

Let's quickly go through some important concepts related to granting and revoking permissions:

- The supported grant operations are `Encrypt`, `Decrypt`, `GenerateDataKey`, `GenerateDataKeyWithoutPlaintext`, `ReEncryptFrom`, `ReEncryptTo`, `CreateGrant`, `RetireGrant`, and `DescribeKey`.
- We can use the `encrypt` subcommand of the AWS KMS API to convert plain text into ciphertext with the help of a key.
- We can use the `decrypt` subcommand of AWS KMS API to convert ciphertext into plain text with the help of the same key that was used for encryption.
- We can use the `re-encrypt` subcommand of the AWS KMS API to decrypt and re-encrypt data on the server side with a new CMK without exposing the plain text on the client side. This subcommand can also be used to change the encryption context of a ciphertext.
- Encryption context is an optional additional set of key-value pairs that form an additional authentication check. The same encryption context that is used for encryption needs to be used for decryption and re-encryption. Since the encryption context is not a secret, it will appear in plain text within AWS CloudTrail logs, making it useful for monitoring and auditing cryptographic operations.
- Grants are an alternative to key policies.
- Within the same account, we can use the key ID or key ARN with the `create-grant` subcommand. For users in other accounts, the ARN needs to be specified.
- The `create-grant` subcommand has a `constraints` parameter that accepts an encryption context.
- When we create grants, the permissions may not be reflected immediately due to the eventual consistency model followed by AWS. By using the grant tokens that are received from the `create-grant` subcommand in further requests, we can avoid any delays due to eventual consistency.
- The `list-grants` subcommand is used to list all the grants for a key and provides the additional `starting-token`, `page-size`, and `max-items` parameters for paginating the result.
- The AWS CLI pagination parameters, `starting-token`, `page-size`, and `max-items`, have the following functions:
  - The `max-items` parameter states the maximum number of items that need to be returned by the API.
  - If there are more results from the API calls than specified by `max-items`, then `NextToken` is provided in the response, which needs to be passed as `starting-token` in the next request.
  - The `page-size` parameter specifies the maximum number of elements to retrieve in a single API call. For example, if `page-size` is 10 and `max-items` is 100, 10 API calls will be made in the background and then 100 items will be returned.

- The `revoke-grant` subcommand can be run by the root user of the account that created it, `RetiringPrincipal` of the grant, or `GranteePrincipal` if they've been given the grant for the `RetireGrant` operation.
- The AWS documentation recommends that, when cleaning up, we retire a grant when we're done using it using the `retire-grant` subcommand. However, we should revoke a grant using the `revoke-grant` subcommand when we intend to actively deny operations that depend on it.
- The `list-retirable-grants` subcommand can be used to list all grants with the specified `RetiringPrincipal`.
- The `list-retirable-grants` subcommand provides the `limit` and `marker` parameters to limit the retrievable grants that need to be returned. Here, `limit` is the maximum number of items that need to be returned, while `marker` is the value of `NextMarker` that is returned with the previous request when more items than what's specified by the `limit` parameter need to be returned.

## See also

To understand the basics of grants in AWS KMS, refer to this blog post: <https://www.cloudericks.com/blog/understanding-grants-in-aws-kms>.

## Using key policies with conditional keys

In this recipe, we will learn how to use **key policies**, especially with conditions. Resource-based policies for KMS keys are called key policies. When managing access to KMS resources, we can use key policies alone, or we can use IAM policies and grants along with key policies. Unlike other resource-based policies such as bucket policies, which are not mandatory, key policies are mandatory to manage and use keys. When a key is created, a default key policy is created by AWS, as we saw in the *Creating keys in KMS* recipe.

## Getting ready

We'll need the following to complete this recipe:

- A working AWS account, `awssecb-sandbox-1`, and a user, `awssecbadmin1`, as described in the *Technical requirements* section.
- An S3 bucket created in the `us-east-1` region. I will use a bucket called `awssecuritykmsbucket`.

## How to do it...

We can demonstrate the use of key policies with condition keys as follows:

1. Create a key with the default configuration from the console, as follows:
  - A. Go to **Key Management Service (KMS)** in the Management Console.
  - B. Select **Customer-managed keys** from the left sidebar and click **Create key**.

- C. In the **Configure key** pane, under **Key type**, select **Symmetric**. Then, under **Key usage**, select **Encrypt and decrypt**. Click **Next**.
- D. Provide **Alias** and **Description** values, leave the other selections as-is, and click **Next**.
- E. On the next screen, do not add any **Key administrators**. Instead, simply click **Next**.
- F. In the **Define key usage permissions** pane, do not add any **Key users** either. Simply click **Next**.
- G. At this point, we should see the policy, as shown in the following figure. Review and click **Finish**:

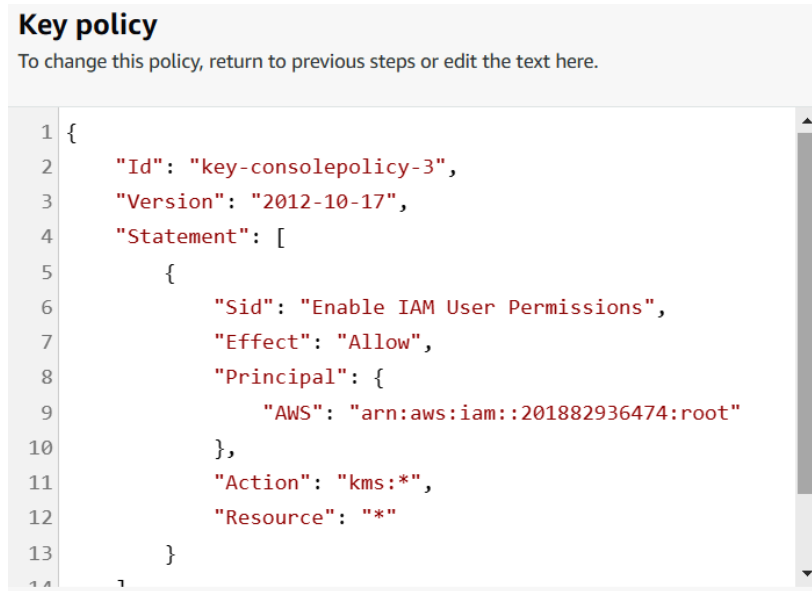


Figure 3.15 – The key policy for review

2. We can add this KMS key as the encryption key for an S3 bucket in the same region as follows:
  - A. Go to the **Properties** tab of our S3 bucket.
  - B. Scroll down to **Default encryption** and click **Edit**.
  - C. In the **Default encryption** pane, under **Encryption type**, select **Server-side encryption with AWS Key Management Service keys (SSE-KMS)**.
  - D. Under **AWS KMS key**, enter the ARN of the key that we created.
  - E. For **Bucket Key**, select **Disable**.

F. Click **Save changes**.

**Default encryption**  
Server-side encryption is automatically applied to new objects stored in this bucket.

Encryption type [Info](#)

- ☐ Server-side encryption with Amazon S3 managed keys (SSE-S3)
- ☒ Server-side encryption with AWS Key Management Service keys (SSE-KMS)
- ☐ Dual-layer server-side encryption with AWS Key Management Service keys (DSSE-KMS)  
Secure your objects with two separate layers of encryption. For details on pricing, see [DSSE-KMS pricing](#) on the **Storage** tab of the [Amazon S3 pricing page](#).

AWS KMS key [Info](#)

- ☐ Choose from your AWS KMS keys
- ☒ Enter AWS KMS key ARN

AWS KMS key ARN

Format (using key id): `arn:aws:kms:<region>:<account-ID>:key/<key-id>`  
(using alias): `arn:aws:kms:<region>:<account-ID>:alias/<alias-name>`

**Bucket Key**  
Using an S3 Bucket Key for SSE-KMS reduces encryption costs by lowering calls to AWS KMS. S3 Bucket Keys aren't supported for DSSE-KMS. [Learn more](#)

- ☒ Disable
- ☐ Enable

**⚠** Changing the default encryption settings might cause in-progress replication and Batch Replication jobs to fail. These jobs might fail because of missing AWS KMS permissions on the IAM role that's specified in the replication configuration. If you change the default encryption settings, make sure that this IAM role has the necessary AWS KMS permissions. [Learn more](#)

Figure 3.16 – Adding an encryption key for an S3 bucket

- Upload a file to S3 and verify that encryption and decryption are working:
  - Upload a file into the S3 bucket.
  - Click on that file and click **Open**. We should be able to view the file's contents.

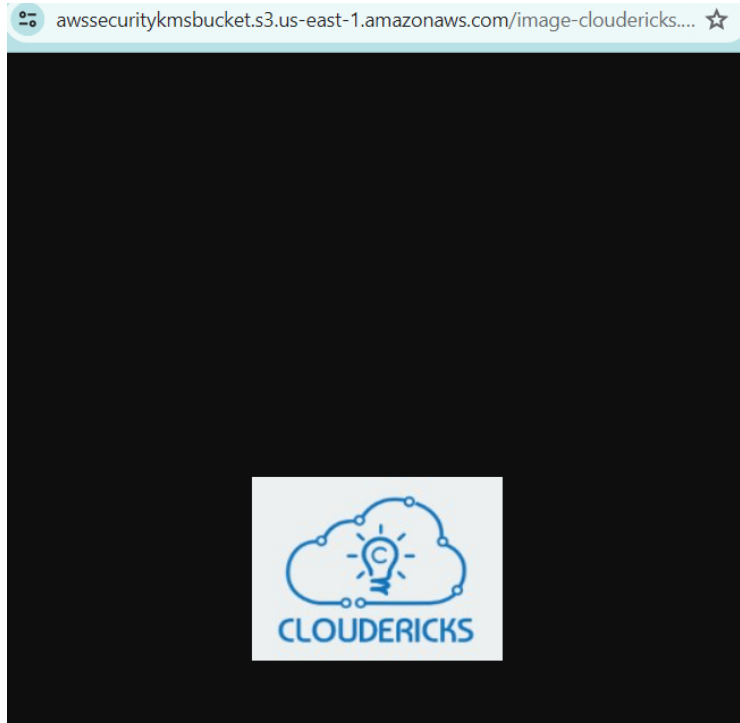


Figure 3.17 – Viewing the content of the S3 bucket

4. Deny key usages from the S3 service by adding a key policy statement:
  - A. Go to **Key Management Service** in the Management Console.
  - B. Click **Customer-managed keys**.
  - C. Click on the key we need to modify.
  - D. Click on the **Key policy** tab.
  - E. Click **Switch to policy view**.
  - F. Click **Edit**, add the following key policy statement with proper commas, and click **Save changes**.

```
{
  "Effect": "Deny",
  "Principal": {
    "AWS": "arn:aws:iam::135301570106:root"
  },
  "Action": [
    "kms:Encrypt",
    "kms:Decrypt",
```



```

    "kms:ReEncrypt*",
    "kms:GenerateDataKey*",
    "kms:CreateGrant",
    "kms:ListGrants",
    "kms:DescribeKey"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:ViaService": "s3.us-east-1.amazonaws.com"
    }
  }
}

```

5. Go back to the same file we opened in *Step 3* and click **Open** to open the file. We won't be able to view the file now and will get the following error. This has happened because S3 doesn't have permission to use the key to perform decryption:



Figure 3.18 – Error while trying to view the content of the S3 bucket after encryption

If we try to run the URL directly on a browser without clicking **Open** from the S3 console, the file won't be displayed with or without the key condition policy since we're using SSE-KMS encryption.

## How it works...

In this recipe, we created a key with default permissions and tried encrypting and decrypting files in the S3 bucket with that key. We were able to successfully encrypt and decrypt. Then, we added an explicit Deny for the S3 service using the `kms:ViaService` condition key and tried decrypting the same file again. This time, we weren't able to decrypt.

As we saw in *Step 1* of this recipe, the default key policy gives full permission to the owner account's root user and enables the IAM policies that are required to access the KMS key. It also allows key administrators to administrate the KMS key and key users to use the KMS key. Also, we need to specify the region for the S3 service while using it within the `ViaService` API. I used `us-east-1` since my bucket is present in `us-east-1`.

In our key policy JSON, we used the following elements:

- **Effect**: Specifies whether to allow or deny permissions.
- **Principal**: Specifies who gets the permissions. Allowed values include AWS accounts (root), IAM users, IAM roles, and supported AWS services.
- **Action**: Specifies the operations (for example, `kms:Encrypt`) to allow or deny.
- **Resource**: Specifies the resource to apply the policy. We specified `*` to denote all the resources.
- **Condition**: Used to specify any condition for the key policy to take effect. This is an optional element.

We can also specify an optional `Sid` parameter. **Sid** stands for **statement identifier** and can contain a string value that describes our policy.

## There's more...

Let's quickly go through some important concepts about using key policies:

- To manage access to KMS resources, we can use key policies alone, or we can use IAM policies and grants along with key policies.
- To allow access to KMS keys, we always need to use key policies, either alone or along with IAM policies or grants.
- The primary resource within KMS is the KMS key.
- A KMS key's ARN has the following form: `arn:aws:kms:<region>:<account ID>:key/<key ID>`.
- Some KMS operations also allow the use of an alias as a resource. An alias ARN has the following form: `arn:aws:kms:<region>:<account ID>:alias/<alias name>`.
- Any user, including the root user, can access the KMS key, but only if the key policy allows it.
- The default key policy when a KMS key is created from the Management Console gives full permission to the owner's account root user and also enables IAM policies that are required to access the KMS key. It will also allow key administrators to administrate the KMS key and key users to use the KMS key. The default key policy when a KMS key is created programmatically gives full permission to the owner's account root user. It also enables IAM policies that are required to access the KMS key.

- When we add key administrators or key users, they are added to the policy document statement with the required permissions. We saw the complete list of permissions for both key administrators and key users in the *Creating keys in KMS* recipe.
- AWS has added a wildcard to some of the default permissions, such as `kms:Create*`, `kms:Describe*`, and others, so that if AWS creates a new action that starts with the same prefix, the administrators or users will get those permissions automatically.
- AWS provides global condition keys, as well as service-specific keys.
- The global condition keys include `aws:PrincipalTag`, `aws:PrincipalType`, `aws:RequestTag`, `aws:SourceIp`, `aws:SourceVpc`, `aws:SourceVpce`, `aws:TagKeys`, `aws:TokenIssueTime`, `aws:userid`, and `aws:username`.
- The AWS KMS condition keys include `kms:BypassPolicyLockoutSafetyCheck`, `kms:CallerAccount`, `kms:EncryptionContext`, `kms:EncryptionContext-Keys`, `kms:ExpirationModel`, `kms:GrantConstraintType`, `kms:GrantIsForAWSResource`, `kms:GrantOperations`, `kms:GranteePrincipal`, `kms:Key-Origin`, `kms:ReEncryptOnSameKey`, `kms:RetiringPrincipal`, `kms:ValidTo`, `kms:ViaService`, `kms:WrappingAlgorithm`, and `kms:WrappingKeySpec`.

## See also

- You can read more about AWS KMS key policies here: <https://www.cloudericks.com/blog/understanding-key-policies-in-aws-kms>.
- You can read about the differences between key policies and grants here: <https://www.cloudericks.com/blog/understanding-aws-kms-key-policies-vs-grants>.
- You can read more about AWS KMS condition keys here: <https://www.cloudericks.com/blog/understanding-aws-kms-condition-keys>.
- You can read more about policies and permissions within AWS here: <https://www.cloudericks.com/blog/demystifying-aws-policies-and-permissions>.

## Sharing customer-managed keys across accounts

In this recipe, we will learn how to use a KMS key from one account in another account.

### Getting ready

We'll need the following to complete this recipe:

- Two working AWS accounts. I will be using the accounts we created in *Chapter 1*, with the account ID of the first account being 135301570106 and the account ID of the second account being 380701114427.

- A user from account 2 with **AdministratorAccess** permission. This could be an IAM user or an IAM Identity Center user. The CLI profile should be configured for this user. I will be calling the CLI profile `Adminuserprofile`.
- Another user without administrator permissions in account 2. The CLI profile should be configured for this user. I will be calling this user and its CLI profile `Testuserprofile`.

## How to do it...

First, we will create a new KMS key in the first account. After that, we will provide permission to use it from the second account. Finally, we will test the first account from the second account using that KMS key.

### *Creating a key and giving permission to the other account*

In this section, we will create a key in account 1 with key usage permission to account 2. Let's get started:

1. Log in to the AWS Management Console and go to **Key Management Service**.
2. Click on **Customer managed keys** from the left sidebar and click on **Create key** at the top right of the page.
3. In the **Configure key** pane, for **Key type**, select **Symmetric**, and for **Key usage**, select **Encrypt and decrypt**. Then, click **Next**.
4. In the **Add labels** pane, provide **Alias** and **Description** values. Click **Next**.
5. On the **Define key administrative permissions** screen, we can add any **Key administrators** if we want to and select the checkbox for **Allow key administrators to delete this key**. Click **Next**.
6. On the **Define key usage permissions** screen, scroll down to the **Other AWS accounts** section and click **Add another AWS account**, as shown in the following figure:

**Other AWS accounts**

Specify the AWS accounts that can use this key. Administrators of the accounts you specify are responsible for managing the permissions that allow their IAM users and roles to use this key. [Learn more](#)

**Add another AWS account**

**Cancel** **Previous** **Next**

Figure 3.19 – The Other AWS accounts section

7. Enter the account ID of the second AWS account and click **Next**.

Figure 3.20 – Adding other AWS accounts

8. On the **Review** screen, review the **Key policy** details and click **Finish**. The statement that's been added to the key policy is available within the code files as `key-policy-sharing-keys.json`.
9. Once we get a success message that the key has been created, click **View key** to see the details of the key. The ARN for my newly created key is as follows:

```
arn:aws:kms:us-east-1:201882936474:key/24ce962a-ee5e-411e-9cb6-ebb39e253c
```

In this section, we created a key in account 1 and gave permission to account 2. In the next section, we will use this key in account 2.

### *Using the key as an administrator user from account 2*

Now, let's try using the key as a user with administrator permission from account 2.

Encrypt the data with the profile of an administrator user from account 2 from the CLI using the following command:

```
aws kms encrypt --plaintext "$(echo -n 'hello heartin' | base64)"
--key-id arn:aws:kms:us-east-1:201882936474:key/24ce962a-ee5e-411e-9cb6-ebb39e253c --profile Adminuserprofile
```

We should see the following response:

```
{
  "CiphertextBlob": "AQICAHInb3sCTwkfhX2M9fzY05h5Ex06IGLgWvNvJapLoU3y+AQEgo
d4S2yAJdGak30pxr7NrAAAAbTBrBgkqhkiG9w0BBwagXjBcAgEAMFcGCSqGSib3DQEHATAeBglggh
kgBZQMEAS4wEQQM8fEsbmLxMLw0LGxpAgEQgCqg68pdgL9DQOyopKIhBMCDvmAi/y2i9hRWaFj0x
iEh0pHCnUs6YZEp2EU=",
  "KeyId": "arn:aws:kms:us-east-1:201882936474:key/24ce962a-ee5e-411e-9cb6
-ebbee39e253c",
  "EncryptionAlgorithm": "SYMMETRIC_DEFAULT"
}
```

Figure 3.21 – Response after running the encrypt command for the admin user

In the next section, we will use the key as a non-admin user from account 2.

### *Using the key as a non-admin user from account 2*

Before proceeding, we can verify that the non-admin user doesn't have access to encrypt the data in account 1 by running the command shown in *Step 1* of the previous section while using the non-admin user profile. Now, follow these steps:

1. To use the KMS key as a non-admin user, the root or the admin user of account 2 has to delegate the permission to the non-admin user. To do so, they must follow these steps:
  - A. Go to the **IAM** dashboard of account 2.
  - B. Click on **Policies** on the left pane.
  - C. Click **Create policy**.
  - D. Go to the **JSON** tab.
  - E. Enter the following policy. Make sure you update the **Sid** value and the **Resource** parameters to reflect your resource details:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DelegateCMKAccessFrom201882936474",
      "Effect": "Allow",
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": "arn:aws:kms:us-east-
1:201882936474:key/24ce962aee5e-411e-9cb6-ebbee39e253c"
    }
  ]
}
```

```
    }
  ]
}
```

- F. Click **Next**.
  - G. In the **Review and create** pane, for **Policy details**, provide **Name** and **Description** detail for our policy.
  - H. Click **Create policy**.
  - I. If you're using an IAM user, assign this policy directly to that IAM user or a group that the user is part of. If you have an IAM Identity Center user, you can assign this policy using a custom permission set, as we saw in the *Creating customer-managed policies in IAM Identity Center* recipe of *Chapter 2*.
2. Encrypt the data with the profile of a non-administrator user from account 2 from the CLI using the following command:

```
aws kms encrypt --plaintext "$(echo -n 'hello
heartin' | base64)" --key-id arn:aws:kms:us-east-
1:201882936474:key/24ce962a-ee5e-411e-9cb6-ebbee39e253c
--profile testprofilenopermission
```

We should see the following response:

```
{
  "CiphertextBlob": "AQICAHiNb3sCTwkFhX2M9fzY05h5Ex06IGLgwNvJapLoU3y+AQFlp
5W59Gzz2muQjsoB+80IAAAAbTBrBgkqhkiG9w0BBwagXjBcAgEAMFcGCSqGSIB3DQEHATAeBglg
kgBZQMEAS4wEQQMjwQcoqX9/zR+FL7cAgEQgCqkqscLLdtmWArIFgaXXHJhjPWu05yli1qwJEA6t
OMipwNyvy9qV0DJzn4=",
  "KeyId": "arn:aws:kms:us-east-1:201882936474:key/24ce962a-ee5e-411e-9cb6-
ebbee39e253c",
  "EncryptionAlgorithm": "SYMMETRIC_DEFAULT"
}
```

Figure 3.22 – Response after running the encrypt command for the non-admin user

In this recipe, we created a new KMS key. We can also edit an existing KMS key and add another account's details.

## How it works...

In this recipe, we created a KMS key in account 1 with permissions for account 2. After that, we successfully encrypted data on the other account with an administrator user's profile from CLI. To encrypt using a non-administrator user, the administrator user of account 2 needs to delegate permissions to the user or role that needs access. We did this through an IAM policy.

For more details on the policy document's structure, refer to the *Using key policies with conditional keys* recipe.

## There's more...

In this recipe, we tried encrypting data from the AWS CLI. Even though delegating key permissions across accounts is supported for most integrated services, such as S3 and EC2, their support for selecting the key automatically from the console may not be supported. Check out each service's documentation for more details. If there is a limitation from the Management Console, then we will need to do this through the API by specifying the ARN of the key.

## See also

We've explored many features of AWS KMS within this chapter. There are many applications and integrations for KMS within the AWS cloud, such as S3 encryption, EBS encryption, and more. Including all of those applications and integrations would require a book dedicated to KMS. You may explore more applications and integrations of KMS within the AWS cloud at <https://www.cloudericks.com/blog/applications-of-aws-kms-within-aws-cloud>.

## Creating, initializing, and activating a CloudHSM cluster

In this recipe, we will create an AWS CloudHSM cluster. CloudHSM is a dedicated **hardware security module (HSM)** on the AWS cloud that we can use to generate and use encryption keys. AWS KMS, on the other hand, uses shared HSM. CloudHSM is ideal for scenarios demanding the highest level of isolation and control.

## Getting ready

We'll need the following to complete this recipe:

- A working AWS account, `awsseccb-sandbox-1`, and a user, `awsseccbadmin1`, as described in the *Technical requirements* section.
- Knowledge of VPCs and EC2. If you are new to EC2 or want to refresh the concepts, you may first practice the recipes in *Chapter 5*, and then come back to this recipe.

### Important note

AWS CloudHSM typically incurs higher costs compared to AWS KMS, and it does not offer a free tier option. For those utilizing CloudHSM for educational or learning purposes, it is crucial to promptly delete the resources you've created once you've completed the relevant recipes. This proactive step helps prevent any unexpected charges.

## How to do it...

First, we need to create a cluster, initialize it, and then activate it from an EC2 instance within the same VPC as the cluster.



## Creating a CloudHSM cluster

In this section, we will create a CloudHSM cluster using the default VPCs, as follows:

1. Go to the **CloudHSM** service in the Management Console.
2. Click on **Create cluster**. If we can't see the **Create cluster** option, we can click on **Clusters** from the left sidebar and then click on **Create cluster** in the right pane.
3. For **VPC**, select the VPC that we want to use or select the default VPC. This cannot be changed later.

### Tip

You may also create a custom VPC with private and public subnets and use them instead of the default VPCs that we're using in this chapter, especially if you're configuring CloudHSM for production. We will be covering VPCs in depth in *Chapter 5*.

4. Select subnets in at least two Availability Zones based on AWS's recommendations. I have chosen three for this recipe in `us-east-1a`, `us-east-1b`, and `us-east-1c`.
5. Scroll down and, under **Cluster source**, select the **Create a new cluster** option. Click **Next** to proceed to the **Back-up retention** step. We also have a **Restore cluster from existing backup** option.
6. In the **Back-up retention** step, under the **Backup retention** screen, for **Backup retention period (in days)**, based on your requirements, enter a period between 7 to 379 days. I have entered 7. Click **Next** to proceed to the **Add tags** step.
7. In the **Add tags** step, optionally add tags to categorize our clusters. Then, click **Next** to proceed to the **Review** step.
8. Review the details on the **Review** screen, including **VPC**, **Availability Zone(s)**, **Days to expire under Back-up retention policy**, and **Tags**. We'll also see a warning stating that we cannot change VPC or subnets after cluster creation.
9. After the review, click **Create cluster**. We should immediately see a message stating that the cluster is being created and that its state is **Creation in progress**. It may take a few minutes for cluster creation to complete. If the cluster creation operation is successful, we should see a screen with a success message and the state will be **Uninitialised**.

In this section, we created a CloudHSM cluster. Next, we need to initialize and activate the cluster.

## Initializing the cluster and creating our first HSM

We can initialize our cluster and create our first HSM by following these steps:

1. Select our cluster and click on **Initialize** from the **Actions** dropdown. Assuming we haven't created any HSM yet, this will take us to the creation wizard for our first HSM.

2. In the **Create HSM in the cluster** step, select one of the Availability Zones we selected while creating the cluster and click **Create**.

HSM creation can take some time. Once this process is successful, we should be taken to the **Download certificate signing request** step and see a success message, similar to the one shown in the following figure:

✓ Your first HSM hsm-bxgvhxij4z has been created in cluster cluster-2i67czpb6yy

Proceed with initialisation

[CloudHSM](#) > [Clusters](#) > [cluster-2i67czpb6yy](#) > Initialise

Step 2 of 3

## Download certificate signing request

### Certificate signing request

To initialise the cluster, you must download a certificate signing request (CSR) and then [sign it](#).

Cluster CSR

### Cluster verification certificate

Optionally, you may wish to download the HSM certificate below which generated this Cluster CSR and [verify its authenticity](#).

HSM certificate

AWS hardware certificate

Manufacturer hardware certificate

Cancel

Previous

Next

Figure 3.23 – Download certificate signing request

3. Click the **Cluster CSR** button to download the **certificate signing request (CSR)**. Optionally, as shown in *Figure 3.23*, we can verify the identity and authenticity of the cluster's HSM by following the *There's more...* section and the *See also* section of this recipe.
4. Click **Next** to proceed to the **Upload certificates** step.
5. Generate a private key by running the `openssl` command on your local machine, like so:

```
openssl genrsa -aes256 -out customerCA.key 2048
```

When prompted, enter and re-enter a passphrase in 4 to 1,023 characters.

6. As this is for testing and development purposes, we can create a self-signed certificate using our private key by running the following command:

```
openssl req -new -x509 -days 3652 -key customerCA.key -out customerCA.crt
```

Answer the prompt questions while providing values similar to the ones shown in the following screenshot:

```
$ openssl req -new -x509 -days 3652 -key customerCA.key -out customerCA.crt
Enter pass phrase for customerCA.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) []:IN
State or Province Name (full name) []:KARNATAKA
Locality Name (eg, city) []:BANGALORE
Organization Name (eg, company) []:TRAINSO
Organizational Unit Name (eg, section) []:CLOUDERICKS
Common Name (eg, fully qualified host name) []:cloudericks.com
Email Address []:heartin@cloudericks.com
```

Figure 3.24 – Generating a self-signed certificate

7. Sign the cluster CSR using the OpenSSL command syntax that's available from the AWS documentation. Remember to mention the exact name of the `csr` file you downloaded from the AWS Management Console and run it from the same folder where it is present:

```
openssl x509 -req -days 3652 -in cluster-2i67czpb6yy_ClusterCsr.
csr -CA customerCA.crt -CAkey customerCA.key -CAcreateserial
-out cluster-2i67czpb6yy_ClusterCsr.crt
```

This should create a file named `cluster-2i67czpb6yy_ClusterCsr.crt`.

8. Go back to the **Upload certificates** screen from *Step 5* of this section and upload the signed cluster certificate and signing certificate (issuing certificate):

The screenshot shows the 'Initialise' step for a CloudHSM cluster. The breadcrumb trail is 'CloudHSM > Clusters > cluster-2i67czpb6yy > Initialise'. Below this, it says 'Step 3 of 3' and 'Upload certificates'. The main section is titled 'Signed cluster certificates' and contains instructions: 'Upload the signed cluster certificate and the issuing certificate.' There are two upload sections: 'Cluster certificate' with a file named 'cluster-2i67czpb6yy\_ClusterCsr.crt' and 'Issuing certificate' with a file named 'customerCA.crt'. At the bottom, there are three buttons: 'Cancel', 'Previous', and 'Upload and initialise'.

Figure 3.25 – Upload certificates

9. Click **Upload and initialise**. We should immediately see a message that cluster initialization is now in progress. Once the initialization is done, we should get a message stating that we must set the cluster password before we can use the cluster. We need to do this from an EC2 instance that's been launched in the same VPC as our CloudHSM cluster.
10. Click on **Clusters** on the left sidebar and verify that our cluster's **State** is set to **Initialised**.
11. We also need to go inside our cluster and copy the **ENI IP address** value of our HSM.

In this section, we initialized the cluster and created our first HSM. In the next section, we will launch an EC2 client instance into the same VPC as our cluster and activate the cluster.

### Activating the cluster

In this section, we will launch an EC2 client instance for our HSM from the AWS Management Console and activate the cluster. Follow these steps:

1. First, we need to launch an EC2 client instance and set up the following options:
  - A. Select **Amazon Linux 2023** under **Operating system**.
  - B. Select **t2.micro** under **Instance type**.
  - C. Use the **Create new key pair** link to create a new key pair and save the private key securely. I have named it `ec2forhsm.pem`.

- D. Under **Network settings**, select the same VPC as our cluster, which is the default VPC in my case. The **Auto-assign public IP** option should be set to **Enable**. In the **Firewall (security groups)** subsection, select **Select an existing security group**, and for **Common security groups**, choose the default security group and the security group of our HSM.
- E. Edit the security group to add an inbound rule for SSH (port range 22) for the IP of the system we are connecting from by selecting the **My IP** option.
2. Copy the self-signed certificate, `customerCA.crt`, into the EC2 machine. On Mac or Linux, we can use the `scp` command to do this, as follows. Remember to use your EC2 **Public IPv4 DNS** value instead of mine, which is `ec2-user@ec2-54-172-128-95.compute-1.amazonaws.com`:

```
scp -i ec2forhsm.pem customerCA.crt ec2-user@ec2-54-172-128-95.compute-1.amazonaws.com:/home/ec2-user
```

3. SSH into the EC2 instance. The exact command or steps may differ between operating systems. On Mac or Linux, we can use the following command. If you face any issues with the permission key permissions, you can run the `chmod 400 ec2forhsm.pem` command to resolve this:

```
ssh -i "ec2forhsm.pem" ec2-user@ec2-54-172-128-95.compute-1.amazonaws.com
```

4. Update the operating system by running the `sudo yum update` command.
5. Run the following command to get the latest CloudHSM client RPM:

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Amzn2023/cloudhsm-cli-latest.amzn2023.x86_64.rpm
```

6. Run the following command to install the CloudHSM client:

```
sudo yum install ./cloudhsm-cli-latest.amzn2023.x86_64.rpm
```

7. Add the **ENI IP address** value of our HSM to the configuration. We took note of it in *Step 12* of the previous section, *Initializing the cluster and creating our first HSM*. Remember to replace my IP with your IP:

```
sudo /opt/cloudhsm/bin/configure -a 172.31.33.71
```

8. Copy the self-signed certificate, `customerCA.crt`, into `/opt/cloudhsm/etc/`:

```
sudo cp /home/ec2-user/customerCA.crt /opt/cloudhsm/etc/
```

9. Run the CloudHSM CLI in interactive mode:

```
/opt/cloudhsm/bin/cloudhsm-cli interactive
```

10. Use the `cluster activate` command to set the initial admin password:

```
[aws-cloudhsm > cluster activate
[Enter password:
[Confirm password:
{
  "error_code": 0,
  "data": "Cluster activation successful"
}
```

Figure 3.26 – Activate CloudHSM cluster

11. Use the `user list` command to verify that the user list:

```
[aws-cloudhsm > user list
{
  "error_code": 0,
  "data": {
    "users": [
      {
        "username": "admin",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "quorum": [],
        "cluster-coverage": "full"
      },
      {
        "username": "app_user",
        "role": "internal(APPLIANCE_USER)",
        "locked": "false",
        "mfa": [],
        "quorum": [],
        "cluster-coverage": "full"
      }
    ]
  }
}
```

Figure 3.27 – List Users in cluster

12. We can quit the utility using the `quit` command.
13. Go back to our cluster within the console and verify that we can't see a display message stating that the cluster can't be used until the password is updated.

In this section, we activated our HSML cluster. HSM clusters don't fall under the free tier. Therefore, if you created this cluster for development or testing purposes, remember to delete the HSM and the cluster to avoid incurring unexpected charges.

## How it works...

In this recipe, we created a CloudHSM cluster, initialized it, and then created our first HSM within it. I used the default VPC for convenience. For practical use cases, you should install HSM into a private subnet within a custom VPC. We will look at VPCs in detail in *Chapter 5*.

Before initializing the cluster, we need to download a CSR and sign it. We used a self-signed certificate while following these steps:

1. Create a private key using OpenSSL.
2. Use the private key to create a self-signed signing certificate (issuing certificate).
3. Use the self-signed signing certificate (issuing certificate) to sign the CSR we downloaded from the AWS Management Console.
4. Finally, upload both our signed CSR certificate and the self-signed issuing certificate to AWS.

For practical use cases, a certificate authority such as Verisign should sign it to create a signed certificate. For development and testing purposes, we can use a self-signed certificate to sign it using OpenSSL, as we saw in this recipe.

First, we logged in to the system as a user with the PRECO role, which is a temporary user role that exists on the first HSM in our cluster. After we changed the default password for this user, we observed that its type changed from PRECO to CO. A user of the AU type was also present.

With CloudHSM, we have four main user types:

- **Precrypto Officer (PRECO)**
- **Crypto Officer (CO)**
- **Crypto User (CU)**
- **Appliance User (AU)**

PRECO is a user role that is created by AWS that we can use until we update the password. Once updated, the user's type is changed to CO. We can create more users with the CO role. The first CO user is referred to as the primary PCO. A CO is responsible for managing users. A CU is responsible for managing keys, including creating, deleting, sharing, importing, and exporting them. A CU is also responsible for cryptographic operations such as encryption, decryption, signing, verifying, and more. Finally, an AU is a limited permission user that is generally used by AWS for cloning and synchronization activities.

---

## There's more...

We can verify the HSM's identity using the certificates that we can download from AWS. At the time of writing, six certificates can be downloaded from CloudHSM and verified: Cluster CSR, HSM certificate, AWS hardware certificate, Manufacturer hardware certificate, AWS root certificate, and Manufacturer root certificate.

## See also

- You can read more about CloudHSM here: <https://www.cloudericks.com/blog/getting-started-with-cloud-hsm>.
- You can learn more about the similarities and differences between CloudHSM and KMS here: <https://www.cloudericks.com/blog/aws-cloudhsm-vs-aws-kms>.





# 4

## Securing Data on S3 with Policies and Techniques

In this chapter, we will delve into securing **Amazon Simple Storage Service (S3)**, an **object store** on the AWS platform. Object storage, which is distinct from traditional **hierarchical file systems**, operates on a key-value principle whereby each object is stored with a unique key as the identifier. Think of S3 as a place where each piece of data has a special name that helps you find it, unlike a regular file system that sorts things into folders.

We have already talked about IAM policies and how to use them for securing data in S3 within the *Creating a customer-managed IAM policy* recipe in *Chapter 2*. We will extend that knowledge and focus on making S3 data-safe using **Access Control Lists (ACLs)** and **bucket policies**. We will also look at other important ways to keep our S3 data secure, such as using S3 features such as **presigned URLs**, **encryption**, **versioning**, and **replication**.

This chapter includes the following recipes:

- Creating an S3 bucket policy
- Working with S3 ACLs
- Creating S3 presigned URLs
- Protecting files with S3 versioning and object locking
- Encrypting data on S3

## Technical requirements

Before diving into the recipes of this chapter, we need to ensure that we have the following knowledge and requirements in place:

- Due to AWS's cloud-based nature, a stable internet connection is essential for accessing and managing services.
- We need an active AWS account to complete most of the recipes within this chapter, and for some recipes, we will need more than one AWS account (as mentioned in the respective recipes).
- A basic understanding of AWS's core concept and a working knowledge of the **AWS Management Console**, **AWS CLI**, and particularly Amazon S3 will benefit us. If you are new to the S3 service, you can refer to the following URL for getting started with S3: <https://www.cloudericks.com/blog/getting-started-with-amazon-s3-on-aws-cloud>
- Knowledge of **IAM Identity Center** from *Chapters 1 and 2* is helpful if we are managing users with IAM Identity Center instead of using IAM directly. If we are planning to use IAM users, we need to be familiar with IAM.
- To execute AWS CLI commands, we need to install **AWS CLI v2**. We also need to configure it using the AWS IAM Identity Center as needed for each recipe if we are managing users with IAM Identity Center. Alternatively, if we are using IAM users for any recipe, we need to configure the CLI profiles using their **access key** and **secret access key**.

### Important note

While it's possible to manage users directly with IAM, AWS now recommends using IAM Identity Center (formerly AWS SSO) whenever possible for enhanced security and convenience.

The code files for this book are available at <https://github.com/PacktPublishing/AWS-Security-Cookbook-Second-Edition>. The code files for this chapter are available at <https://github.com/PacktPublishing/AWS-Security-Cookbook-Second-Edition/tree/main/Chapter04>.

## Creating an S3 bucket policy

Previously, we had explored creating IAM policies. IAM policies are generally used for user-level permissions across AWS services, while bucket policies are specific to individual S3 buckets and offer more granular control at the bucket level. For instance, bucket policies uniquely enable granting access to anonymous users, enforcing **Server-Side Encryption (SSE)** by default, and restricting access based on source IP or VPC.

In this recipe, we will first create a bucket policy from the Management Console by generating a policy using the Policy Generator that allows `ListBucket` and `GetObject` actions to everyone. Then, we will create a bucket policy from AWS CLI that provides access to a specific IAM user. You can use both the policies from either the Management Console or the AWS CLI; however, I wanted to show how it is done with both approaches. I will also provide examples of principal types in the *There's more* section of this recipe.

## Getting ready

We need the following to successfully complete this recipe:

- A working AWS account is essential. I will be using the `awssecbb-sandbox-1` account that we created in *Chapter 1*. However, I will not be using any features of the AWS Organizations or the IAM Identity Center.
- An `awssecbb_admin1` user with **AdministratorAccess** permission is also important.
- We also need an S3 bucket and a file in it. I will use a bucket named `awssecbbbucketpolicy` with a file named `image-cloudericks.png`. Replace them with your bucket name and filename. The S3 bucket should be configured with **Block all public access** unchecked, especially for the settings related to bucket policies. We can do this while creating a bucket, as shown in *Figure 4.1*. For the rest of the settings, keep the defaults as they are, which, at the time of writing this book, are as follows:
  - **ACLs disabled (recommended)** is selected
  - **Bucket Versioning** is set to **Disable**
  - **Default encryption** is set to **Server-side encryption with Amazon S3 managed keys (SSE-S3)**
  - **Bucket Key** is set to **Enable**
  - **Object Lock** (which is found under **Advanced settings**) is set to **Disable**


☐ **Block all public access**  
Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

☒ **Block public access to buckets and objects granted through *new* access control lists (ACLs)**  
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.

☒ **Block public access to buckets and objects granted through *any* access control lists (ACLs)**  
S3 will ignore all ACLs that grant public access to buckets and objects.

☐ **Block public access to buckets and objects granted through *new* public bucket or access point policies**  
S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.

☐ **Block public and cross-account access to buckets and objects through *any* public bucket or access point policies**  
S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

 **Turning off block all public access might result in this bucket and the objects within becoming public**  
AWS recommends that you turn on block all public access, unless public access is required for specific and verified use cases such as static website hosting.

☒ I acknowledge that the current settings might result in this bucket and the objects within becoming public.

Figure 4.1 – Unchecking settings for bucket policies

- Verify that your bucket does not allow listing for everyone by going to the bucket from the browser using the `https://<bucket-name>.s3.amazonaws.com` bucket URL, replacing `<bucket-name>` with your bucket name (for example, `https://seccbbucket.s3.amazonaws.com/`).

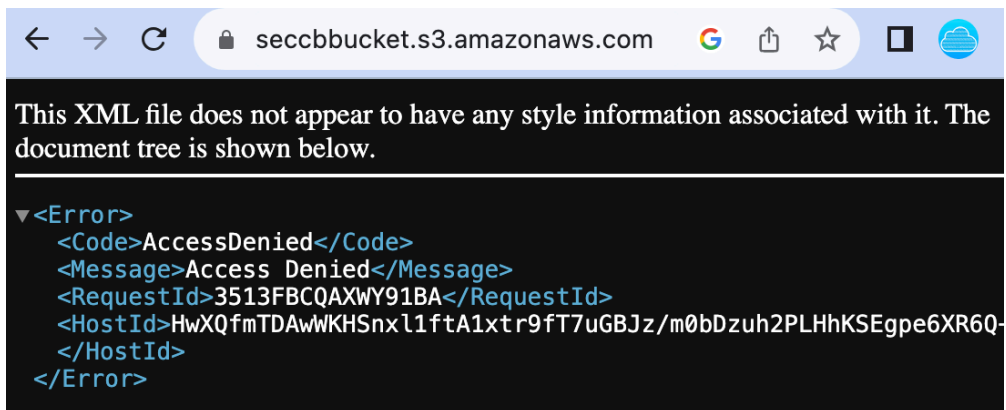


Figure 4.2 – Verifying access for the bucket before starting the recipe

- For using an IAM user as the principal, as outlined in the *Granting ListBucket access for an IAM user from the CLI* section, we need an IAM user with no permissions called `awsseccb_user1`.
- We will also need an environment setup for executing CLI commands with two CLI profiles, `AwsSecCbAdmin` and `AwsSecCbUser`, for the `awsseccb_admin1` and `awsseccb_user1` users, respectively, following the *Technical requirements* section of this chapter if we want to follow the steps involving CLI commands.

Next, we will use bucket policies to give permission to everyone to list the contents of our bucket and then retry this step.

## How to do it...

We will first generate a policy from the console using the policy generator. Later, we will execute the policy from the CLI.

### *Granting ListBucket and GetObject access from the console*

We can give public access to list the contents of a bucket as follows:

1. Go to the **S3** service in the console, click on our bucket's name, go to the **Permissions** tab, and then go to **Bucket policy**.
2. Click on **Edit**, then click on **Policy generator** on the upper-right side.
3. Within the **AWS Policy Generator** page, select or enter data as follows:
  - I. For **Select Type of Policy**, select **S3 Bucket Policy**.
  - II. For **Effect**, select **Allow**.
  - III. For **Principal**, enter the `*` value.
  - IV. For **AWS Service**, **Amazon S3** will be selected.
  - V. For **Actions**, select **ListBucket**.
  - VI. For **Amazon Resource Name (ARN)**, copy and paste the Bucket ARN (e.g., `arn:aws:s3:::seccbbucket`) from the **Edit bucket policy** page.

→ ↻ 🔒 awspolicygen.s3.amazonaws.com/policygen.html

## AWS Policy Generator

The AWS Policy Generator is a tool that enables you to create policies that control access to [Amazon Web Ser](#). For more information about creating policies, see [key concepts in Using AWS Identity and Access Manager](#).

### Step 1: Select Policy Type

A Policy is a container for permissions. The different types of policies you can create are an [IAM Policy](#), an [S3 VPC Endpoint Policy](#), and an [SQS Queue Policy](#).

**Select Type of Policy** S3 Bucket Policy ▾

### Step 2: Add Statement(s)

A statement is the formal description of a single permission. See [a description of elements](#) that you can use i

**Effect** ☒ Allow ☐ Deny

**Principal**   
Use a comma to separate multiple values.

**AWS Service** Amazon S3 ▾ ☐  
Use multiple statements to add permissions for more than one service.

**Actions** 1 Action(s) Selected ▾ ☐ All Actions ('\*')

**Amazon Resource Name (ARN)** arn:aws:s3:::seccbucket  
ARN should follow the following format: arn:aws:s3:::\${BucketName}/\${KeyName}.  
Use a comma to separate multiple values.

[Add Conditions \(Optional\)](#)

**Add Statement**

Figure 4.3 – The AWS Policy Generator page

4. Click **Add Statement**.
5. Click **Generate Policy**.

The generated policy should look similar to the following.

## Policy JSON Document

Click below to edit. To save the policy, copy the text below to a text editor. Changes made below will **not be reflected** in the policy generator tool.

```
{
  "Id": "Policy1700759457658",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1700758997960",
      "Action": [
        "s3:ListBucket"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::seccbbucket",
      "Principal": "*"
    }
  ]
}
```

Figure 4.4 – The bucket policy

6. Copy the policy into the **Bucket policy** editor on the **Edit bucket policy** page, then click **Save changes**.
7. In a browser, provide the bucket URL (e.g., `https://seccbbucket.s3.amazonaws.com`). The contents of the bucket should now be listed:

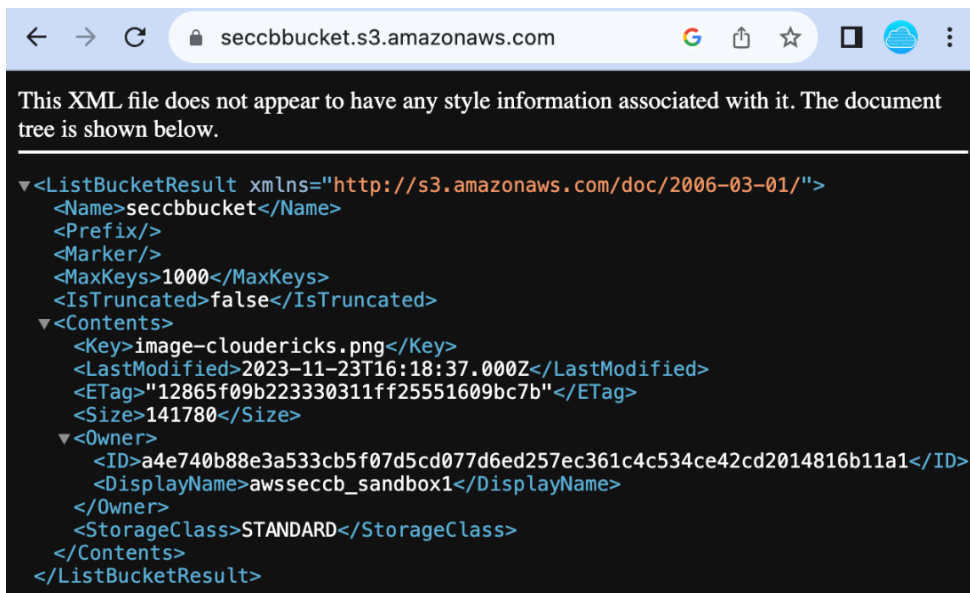


Figure 4.5 – Listing the contents of the bucket



8. In the bucket policy editor, change `s3:ListBucket` to `s3:GetObject` and add `/*` at the end of the resource (such as `"Resource": "arn:aws:s3:::seccbbucket/*"`), then click **Save changes**.
9. Access the object URL from the bucket and paste it into the browser. We should be able to successfully retrieve the object now:



Figure 4.6 – Accessing the image from the bucket

If we change the resource to `arn:aws:s3:::seccbbucket/*` without an object operation such as `s3:GetObject`, we will get an error stating that the action does not apply to any resource. This is because, when we add any prefix (such as `*`) to the bucket, it is considered an object operation; otherwise, it is considered a bucket-level operation.

Next, we will see how we can grant `ListBucket` access to an IAM user from the CLI.

### ***Granting ListBucket Access for an IAM user from the CLI***

In the previous section, we used a policy that allows access to everyone by specifying **Principal** as `*`. In this section, we will use a bucket policy that provides access to a specific IAM user to demonstrate creating a bucket policy from the CLI. Let us get started:

1. If you are following along from the previous section, delete that policy and make sure that you do not have access to list the bucket.
2. Create a bucket policy to allow our `awsseccb_user1` user to access it and save it as `bucket-policy-allow-list.json`:

```
{
  "Id": "Policy1560416549842",
  "Version": "2012-10-17",
  "Statement": [
```

```
{
  "Sid": "ListAllBuckets",
  "Action": [
    "s3:ListBucket"
  ],
  "Effect": "Allow",
  "Resource": "arn:aws:s3:::awsseccbbucketpolicy",
  "Principal": {
    "AWS": "arn:aws:iam::207849759248:user/awsseccb_user1"
  }
}
```

Remember to replace my S3 bucket name, `awsseccbbucketpolicy`, with your bucket name and my AWS account ID, `207849759248`, with your AWS account ID within the given policy.

3. Attach the policy to the bucket:

```
aws s3api put-bucket-policy --bucket awsseccbbucketpolicy
--policy file://bucket-policy-allow-list.json --profile
AwsSecCbAdmin
```

4. List the contents of the bucket using the `AwsSecCbUser` profile created for the `awsseccb_user1` user, using the `aws s3 ls bucketname` command as shown in the following screenshot:

```
$aws s3 ls awsseccbbucketpolicy --profile AwsSecCbUser
2023-11-23 21:40:37      141780 image-cloudericks.png
```

Figure 4.7 – Listing the contents of the bucket using CLI

Now that you have seen how to create policies from the console and the CLI, practice more scenarios with each of the available actions and conditions.

## How it works...

In this recipe, we created S3 bucket policies. A bucket policy statement can have the following components: **Sid**, **Principal**, **Effect**, **Action**, **Resource**, and **Condition**. All of these except **Principal** are the same as an IAM policy and we explored them in the *Creating a customer-managed IAM policy* recipe in *Chapter 2*.

The **Principal** component for a bucket policy can be an account, user, role, or everyone (denoted by `*`). It can contain an ARN for a resource (specified using the `ARN` element) or a canonical ID (specified using the `CanonicalUser` element). More details about the principal are present in *There's more* section of this recipe.

Resource, in the case of a bucket policy, is a bucket or object and is denoted using a bucket ARN. The bucket ARN should be in the `arn:aws:s3:::bucket_name` form. An object resource is represented in the `arn:aws:s3:::bucket_name/key_name` form. To denote all objects within a bucket, we can use `arn:aws:s3:::bucket_name/*`. We can denote every resource in every bucket as `arn:aws:s3:::*`.

## There's more...

Bucket policies follow the same JSON document structure as IAM policies but have an additional principal field. The principal is the user or entity for which a policy statement is applicable. There is no principal for an IAM policy as it is attached to an IAM user. The IAM user who executes that policy is the principal in the case of an IAM policy.

Consider the following examples when using Principal in bucket policies:

- A root user can be represented as follows:

```
"Principal": {
  "AWS": "arn:aws:iam::1312086767378:root"
}
```

- An IAM user can be represented as follows:

```
"Principal": {
  "AWS": "arn:aws:iam::312086767378:user/demouser"
}
```

- A federated user can be represented as follows:

```
"Principal": {
  "AWS": "arn:aws:iam::312086767378:user/demouser"
}
```

- An IAM role can be represented as follows:

```
"Principal": {
  "AWS": "arn:aws:iam::your-account-id:role/MyRole"
}
```

- A role session can be represented as follows:

```
"Principal": {
  "AWS": "arn:aws:sts::767883004914:assumed-role/UserRole/MySession"
}
```

- A canonical user ID can be represented as follows:

```
"Principal": {
  "CanonicalUser": "5df5b6014ae606808dcb64208aa09e4f19931b3123
456e152c4dfa52d38bf8fd"
}
```

- An AWS session can be represented as follows:

```
"Principal": {
  "Service": "cloudfront.amazonaws.com"
}
```

- Multiple principals can be represented in an array as follows:

```
"Principal": {
  "AWS": [
    "arn:aws:iam::873506153865:root",
    "arn:aws:iam::671100771477:root"
  ]
}
```

- An anonymous user can be represented as follows:

```
"Principal" : "*"
```

Let's quickly go through some more important details relating to S3 bucket policies:

- The Conditions element is an optional element that allows us to conditionally execute policies. We used Conditions in one of the examples.
- Currently, we have around 50 bucket policy actions, including those that work on an object (for example, `s3:PutObject`), a bucket (for example, `s3:CreateBucket`), or a bucket sub-resource (for example, `PutBucketAcl`).
- The current list of bucket sub-resources with permissions includes `BucketPolicy`, `BucketWebsite`, `AccelerateConfiguration`, `BucketAcl`, `BucketCORS`, `BucketLocation`, `BucketLogging`, `BucketNotification`, `BucketObjectLockConfiguration`, `BucketPolicyStatus`, `BucketPublicAccessBlock`, `BucketRequestPayment`, `BucketTagging`, `BucketVersioning`, `EncryptionConfiguration`, `InventoryConfiguration`, `LifecycleConfiguration`, `MetricsConfiguration`, `ReplicationConfiguration`, and `AnalyticsConfiguration`.
- We cannot specify an IAM group as a principal in an S3 bucket policy. If we add a group instead of a user, we will get an error: `Invalid principal in policy`.

- Here are some S3-specific condition keys available for use in conditions within a policy: `s3:x-amz-acl`, `s3:x-amz-copy-source`, `s3:x-amz-metadatadirective`, `s3:x-amz-server-side-encryption`, `s3:VersionId`, `s3:LocationConstraint`, `s3:delimiter`, `s3:max-keys`, `s3:prefix`, `s3:xamz-server-side-encryption-aws-kms-key-id`, `s3:ExistingObjectTag/`, `s3:RequestObjectTagKeys`, `s3:RequestObjectTag/`, `s3:object-lock-remainingretention-days`, `s3:object-lock-mode`, `s3:object-lock-retain-untildate`, and `s3:object-lock-legal-hold`.

## See also

- We can read about IAM policies in the *Creating a customer-managed IAM policy* recipe from *Chapter 2*. For a detailed comparison of ACLs, bucket policies, and IAM policies, refer to the *There's more* section of the *Working with S3 ACLs* recipe in the current chapter.
- We can find many bucket policy examples at <https://docs.aws.amazon.com/AmazonS3/latest/userguide/example-bucket-policies.html>.

## Working with S3 ACLs

In Amazon S3, ACLs are utilized to manage access to both buckets and objects. As we delve into ACLs, it's crucial to recognize that they are now viewed as a legacy tool within the AWS ecosystem. AWS recommends opting for more current solutions such as IAM and bucket policies, which provide enhanced flexibility and security. Nonetheless, understanding ACLs is beneficial, particularly when dealing with older systems or applications that were developed before the advent of IAM and bucket policies.

In this recipe, we will learn to grant permissions to the public (everyone) to list the files of a bucket using ACLs from the AWS Management Console. We will list more use cases within the *There's more* section of the recipe.

## Getting ready

We need the following to successfully complete this recipe:

- A working AWS account is essential. I will be using the `awssecCb-sandbox-1` account that we created in *Chapter 1*. However, I will not be using any features of the AWS Organizations or the IAM Identity Center.
- An `awssecCb_admin1` user with **AdministratorAccess** permissions.
- We will need an environment setup for executing CLI commands with two CLI profiles, `AwsSecCbAdmin` and `AwsSecCbUser`, for the `awssecCb_admin1` and `awssecCb_user1` users (respectively) following the *Technical requirements* section of this chapter if we want to follow the steps involving CLI commands.

- We will also need an S3 bucket and a file in it. I will be using a bucket named `awsseccbac1demo` with a file named `image-cloudericks.png`. Replace them with your bucket name and filename. The S3 bucket should be configured with ACLs enabled. We can do this while creating a bucket, as shown in *Figure 4.8*. Also, ensure that the **Block all public access** option is unchecked, especially for the settings related to ACLs. We can do this while creating a bucket, as shown in *Figure 4.9*. For the rest of the settings, keep the defaults as they are, which, at the time of writing this book, are as follows:
  - **Bucket Versioning** is set to **Disable**
  - **Default encryption** is set to **Server-side encryption with Amazon S3 managed keys (SSE-S3)**
  - **Bucket Key** is set to **Enable**
  - **Object Lock** (found under **Advanced settings**) is set to **Disable**

While creating an S3 bucket, we can enable ACLs as required for this recipe with the following settings:

### Object Ownership [Info](#)


Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

#### ☐ ACLs disabled (recommended)

All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.

#### ☒ ACLs enabled

Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.

 We recommend disabling ACLs, unless you need to control access for each object individually or to have the object writer own the data they upload. Using a bucket policy instead of ACLs to share data with users outside of your account simplifies permissions management and auditing.

### Object Ownership

#### ☒ Bucket owner preferred

If new objects written to this bucket specify the bucket-owner-full-control canned ACL, they are owned by the bucket owner. Otherwise, they are owned by the object writer.

#### ☐ Object writer

The object writer remains the object owner.




 If you want to enforce object ownership for new objects only, your bucket policy must specify that the bucket-owner-full-control canned ACL is required for object uploads. [Learn more](#) 

Figure 4.8 – Enabling ACLs for Amazon S3 object ownership

Furthermore, we can uncheck the **Block all public access** option as required for this recipe, as shown in the following figure:

☐ **Block all public access**  
Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

- ☐ **Block public access to buckets and objects granted through *new* access control lists (ACLs)**  
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
- ☐ **Block public access to buckets and objects granted through *any* access control lists (ACLs)**  
S3 will ignore all ACLs that grant public access to buckets and objects.
- ☒ **Block public access to buckets and objects granted through *new* public bucket or access point policies**  
S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.
- ☒ **Block public and cross-account access to buckets and objects through *any* public bucket or access point policies**  
S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

 **Turning off block all public access might result in this bucket and the objects within becoming public**  
AWS recommends that you turn on block all public access, unless public access is required for specific and verified use cases such as static website hosting.

☒ I acknowledge that the current settings might result in this bucket and the objects within becoming public.

Figure 4.9 – Unchecking the Block all public access setting for ACLS

Verify that our bucket does not allow listing for everyone by going to the bucket from the browser using the `https://<bucket-name>.s3.amazonaws.com` bucket URL, replacing `<bucket-name>` with our bucket name (e.g., `https://awsseccbacldemo.s3.amazonaws.com`), similar to *Figure 4.2*.

Next, allow everyone to list the bucket's contents.


## How to do it...

Perform the following steps to allow everyone to list the bucket's contents:

1. Go to the **S3** service on the console.
2. Click on our bucket's name (e.g., **awsseccbacldemo**) to go to the bucket's page.

3. Go to the **Permissions** tab of the bucket, scroll down to the **Access control list (ACL)** tab, and click **Edit**.
4. On the **Edit access control list (ACL)** page, under **Everyone (public access)**, select the checkbox for **List** under **Objects**, and select the checkbox next to the **I understand the effects of these changes on my objects and buckets** statement.

Everyone (public access)	<input checked="" type="checkbox"/> <b>List</b>	<input type="checkbox"/> Read
Group:	<input type="checkbox"/> Write	<input type="checkbox"/> Write
<a href="http://acs.amazonaws.com/groups/global/AllUsers">http://acs.amazonaws.com/groups/global/AllUsers</a>		
Authenticated users group (anyone with an AWS account)	<input type="checkbox"/> List	<input type="checkbox"/> Read
Group:	<input type="checkbox"/> Write	<input type="checkbox"/> Write
<a href="http://acs.amazonaws.com/groups/global/AuthenticatedUsers">http://acs.amazonaws.com/groups/global/AuthenticatedUsers</a>		
S3 log delivery group	<input type="checkbox"/> List	<input type="checkbox"/> Read
Group:	<input type="checkbox"/> Write	<input type="checkbox"/> Write
<a href="http://acs.amazonaws.com/groups/s3/LogDelivery">http://acs.amazonaws.com/groups/s3/LogDelivery</a>		

 When you grant access to the Everyone or Authenticated users group grantees, anyone in the world can access the objects in this bucket.

[Learn more](#) 

☒ I understand the effects of these changes on my objects and buckets.

Figure 4.10 – Editing the ACL page

5. Scroll down and click **Save changes**.
6. Access the bucket from the browser using the `https://<bucket-name>.s3.amazonaws.com` link, replacing `<bucket-name>` with your bucket name (e.g., `https://awsseccbacldemo.s3.amazonaws.com/`). You should be able to list the contents of the bucket.



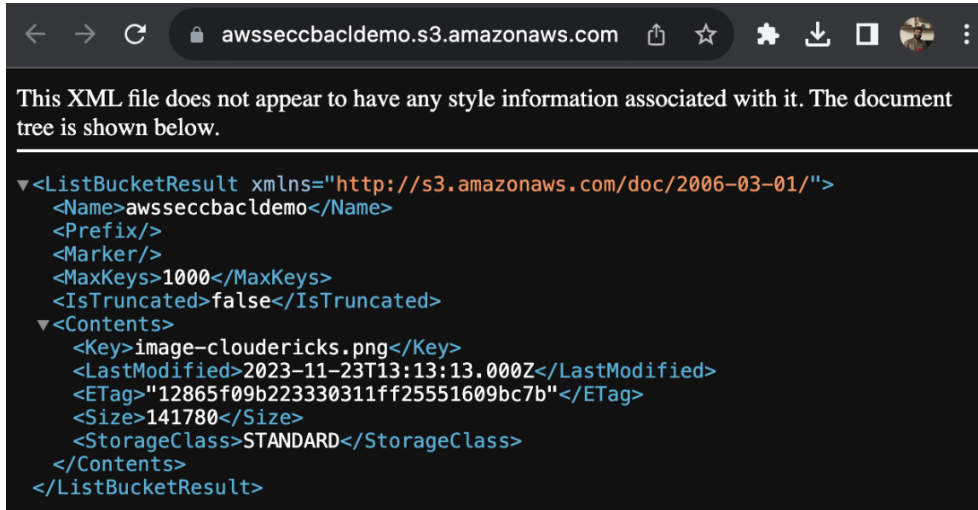


Figure 4.11 – An XML file showing the contents of the bucket

In this edition of the book, we focus on a narrower range of scenarios involving ACLs, unlike the broader coverage of use cases in the previous edition of this book, as ACLs are currently considered a legacy option.

## How it works...

This recipe provided the steps for enabling the public listing of an S3 bucket's contents in AWS. The key step involved modifying the ACL settings via the AWS Management Console. In doing so, we specifically configured the ACL to allow public access for listing the bucket's contents, while ensuring that general read access to the bucket's objects was not granted.

In this recipe, we allowed everyone to list the contents of the bucket using ACLs from the Management Console. We can also do the same from CLI. ACLs provide foundational read/write permissions for buckets, objects, and their ACLs. Specifically, they can grant the following permissions and can be set while working with the AWS CLI:

- **READ:** Allows listing objects in a bucket and reading an object and its metadata
- **WRITE:** Enables the creation, overwriting, or deletion of objects in a bucket; this permission is not applicable to individual objects
- **READ\_ACP:** Grants the ability to read the ACL of a bucket or object
- **WRITE\_ACP:** Permits writing the ACL for a bucket or object
- **FULL\_CONTROL:** Includes all of the previous permissions, offering complete authority over the bucket or object

Prior to implementing READ ACLs for public access, we prepared the bucket by enabling ACLs and ensuring that the **Block all public access** setting was unchecked, specifically for ACL-related options. It's important to note that ACLs, often regarded as a legacy feature in comparison to more recent options such as IAM and bucket policies, are disabled by default. Likewise, public access is blocked by default as a security measure.

## There's more...

Let's delve deeper into some important concepts related to ACLs in Amazon S3:

- **Basic permissions:** ACLs provide foundational read/write permissions for buckets, objects, and their ACLs.
- **Access granting:** ACLs grant access to AWS accounts and predefined groups but cannot deny access. Predefined groups in Amazon S3 ACLs are fixed categories set by AWS that represent broad sets of users, such as **All Users** for the public, **Authenticated Users** for any logged-in AWS user, and **Log Delivery** for services that deliver logs to your bucket. They simplify permission settings without specifying individual user accounts. When specifying grantees, the following methods can be used:
  - **AmazonCustomerByEmail:** Set **Type** to **AmazonCustomerByEmail** and use the canonical ID in the **EmailAddress** field
  - **CanonicalUser:** Set **Type** to **CanonicalUser**; the account's email is provided in the **ID** field
  - **Group:** When **Type** is **Group**, use the URI of the predefined group in the **URI** field
- **Default control:** ACLs, by default, grant full control to the resource owner and no permissions to anyone else.
- **Internal representation:** ACLs are formatted as XML documents, specifying access permissions and grantees.
- **Non-owner object access:** ACLs allow granting access to objects not owned by the bucket owner using canned ACLs such as `bucket-owner-full-control`.
- **Canned ACLs:** These are short-hand ACL permissions that can be used to provide permission for a resource from the command line. Currently, the following canned ACLs are supported: `private`, `public-read`, `public-read-write`, `aws-exec-read`, `authenticated-read`, `bucket-owner-read`, `bucket-owner-full-control`, and `log-delivery-write`.
- **Granular object permissions:** ACLs offer a simpler method for assigning individual permissions to numerous objects compared to bucket policies.
- **Block public access override:** S3's **Block Public Access** settings can override ACLs granting public access.
- **Logging access attempts:** Both successful and denied access attempts can be logged using S3 access logs.

- **No conditions support:** ACLs do not support conditional clauses for granting access, unlike IAM or bucket policies.
- **Management challenges:** Managing ACLs in large environments can be complex.
- **Security best practices:** Regular reviews and audits of ACL settings are crucial for maintaining security.
- **Scope and application differences:** We have the following:
  - **ACLs:** Specified per resource (bucket or object)
  - **Bucket policies:** Applied to entire buckets and can be used to define access based on object prefixes
  - **IAM policies:** Similar in scope to bucket policies but are assigned to IAM users and groups, allowing for more centralized and granular access control

Let's quickly go through some more important concepts related to canned ACLs:

- The `bucket-owner-read` and `bucket-owner-full-control` canned ACLs are only applicable to objects and are ignored if specified while creating a bucket.
- The `log-delivery-write` canned ACL only applies to a bucket.
- With the `aws-exec-read` canned ACL, the owner gets the `FULL_CONTROL` permission and Amazon EC2 gets `READ` access to an **Amazon Machine Image (AMI)** from S3.
- With the `log-delivery-write` canned ACL, the `LogDelivery` group gets `WRITE` and `READ_ACP` permissions for the bucket. This is used for S3 access logging.
- When making an API call, we can specify a canned ACL in our request using the `x-amz-acl` request header.

#### **Important note**

In the case of cross-account access, if a user from account A uploads an object to a bucket in account B (owned by account B), account B will have no access to that object even if it is the bucket owner. Account A can, however, grant permission to the bucket owner while uploading the document using the `bucket-owner-read` or `bucket-owner-full-control` canned ACL.

### ***Comparing ACLs, bucket policies, and IAM policies***

ACLs differ from IAM policies and bucket policies in the following ways:

- ACLs provide only basic read/write permission to buckets, objects, and their ACLs. IAM policies and bucket policies provide more fine-grained permissions than ACLs.
- ACLs can only grant access to AWS accounts and predefined groups. ACLs cannot grant permissions to IAM users. IAM policies and bucket policies can be used to grant access to IAM users.

- ACLs, by default, allow full control to the owner of the resource and nothing to everyone else. Bucket policies and IAM policies are not attached to a resource by default.
- ACLs can only grant permissions. Bucket policies and IAM policies can explicitly deny access.
- ACLs cannot conditionally allow or deny access. Bucket policies and IAM policies can conditionally allow or deny access.
- ACLs are represented internally as XML documents. Bucket policies and IAM policies are represented as JSON documents.

IAM policies differ from ACLs and bucket policies in the following ways:

- IAM policies are user-based and are applied to users. ACLs and bucket policies are resource-based policies and are applied to resources.
- IAM policies can be inline (embedded directly into a user, group, or role) or standalone (they can be attached to any IAM user, group, or role). ACLs and bucket policies are sub-resources of a bucket.
- IAM policies can only give access to an IAM user. Bucket policies and ACLs can be used to provide anonymous access as well as access to a root user.

#### Important note

We can mix ACLs, bucket policies, and IAM policies. All policies are evaluated at the same time if the bucket and user are within the same account.

## See also

- We can learn about IAM policies in the *Creating a Customer Managed IAM Policy* recipe in *Chapter 2*.
- We can learn more about IAM policies and permissions at <https://www.cloudericks.com/blog/demystifying-aws-policies-and-permissions>.

## Creating S3 presigned URLs

We can grant temporary permission to access S3 objects using presigned URLs with an expiry time. In this recipe, we will learn to use presigned URLs. We can do this from the Management Console or AWS CLI or by using an SDK for a programming language such as Python, Java, and so on.

## Getting ready

We will need the following to successfully complete this recipe:

- A working AWS account; I will be using the `awssecCb-sandbox-1` account that we created *Chapter 1*.

- An S3 bucket and a file in it; I will use a bucket named `seccbbucket` with a file named `image-cloudericks.png`.

## How to do it...

We will first create a presigned URL from the console and then from the AWS CLI.

### *Generating a presigned URL from the AWS Console*

We can create a presigned URL from the console and test it as follows:

1. Log in to the AWS Management Console, navigate to the **S3** service, and go into our S3 bucket.
2. Select the object for which we need to create the presigned URL, click on the **Object actions** dropdown, and select **Share with a presigned URL**.
3. Enter the value for **Number of minutes** as 5, as shown in the following figure, and click on **Create presigned URL**.

Share "image-cloudericks.png" with a presigned URL

Presigned URLs are used to grant access to an object for a limited time. [Learn more](#)

Anyone can access the object with this presigned URL until it expires, even if the bucket, and object are private.

Time interval until the presigned URL expires

Using the S3 console, you can share an object with a presigned URL for up to 12 hours or until your session expires. To create a presigned URL with a longer time interval, use the AWS CLI or AWS SDK. Time intervals for presigned URLs can be restricted by your IAM policy.

☒ Minutes

☐ Hours

Number of minutes

Must be a whole number between 1 and 720.

After you create the presigned URL, it's automatically copied to your clipboard.

Cancel

Create presigned URL

Figure 4.12 – Creating a presigned URL

We should get a **Copy presigned URL** notification as follows:

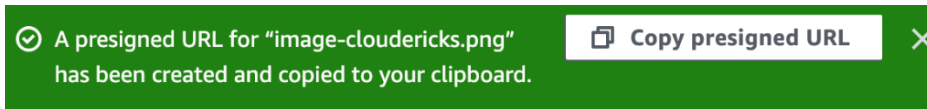


Figure 4.13 – A Copy presigned URL message

4. Copy and paste the URL and run it from a browser within the time mentioned for **Number of minutes**. We should then be able to see our file successfully.

If we run the URL after the specified time, we should get an **AccessDenied** error message similar to the following:

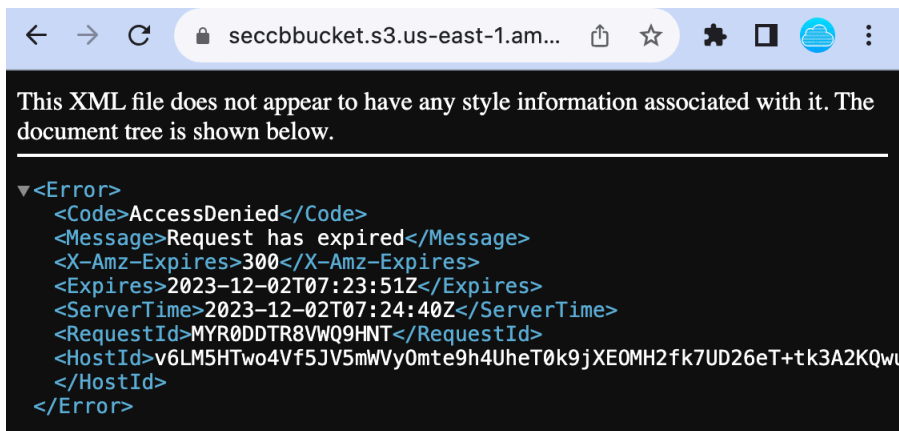


Figure 4.14 – Accessing the presigned URL after expiry

Next, we will look at how to do pre-signing using the CLI.

### ***Generating a presigned URL from the CLI***

We can create a presigned URL from the CLI and test it as follows:

1. Pre-sign a URL from the CLI as follows:

```
aws s3 presign s3://seccbbucket/image-cloudericks.png
--expires-in 100 - profile AwsSecCbAdmin
```

This command will output a presigned URL similar to the following.

```
$aws s3 presign s3://seccbbucket/image-cloudericks.png
--expires-in 100 --profile AwsSecCbAdmin
https://seccbbucket.s3.us-east-1.amazonaws.com/image-cloudericks.png?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=ASIATAZGPUIIGKAGWRRZ%2F20231202%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20231202T073544Z&X-Amz-Expires=100&X-Amz-SignedHeaders=host&X-Amz-Security-Token=IQoJb3JpZ2luX2VjEMD%2F%2F%2F%2F%2F%2F%2F%2F%2FwEaCXVzLWVhc3QtMSJHMEUCIQDvyUkEtrGJPPY56EdtF%2FIOjTs9BFsZ43u3
```

Figure 4.15 – Generating a presigned URL (partial) from CLI

2. Run the URL from a browser (as we did in the previous section) before and after the expiry.

Next, we will explore how to do pre-signing using an SDK such as Python SDK.

## How it works...

In the *Generating a presigned URL from the AWS Console* section, we presigned a URL from the AWS console. In the *Generating a presigned URL from the CLI* section, we presigned a URL from the CLI.

Most APIs related to presigning will accept the following data for generating presigned, timed URLs:

- Bucket and object
- Expiry date and time
- HTTP method
- Security credentials

In this recipe, we specified the bucket, object, and expiry in the code. The HTTP operation was GET. For security credentials, we specified a user profile that has permissions for the operation, which was `get_object` in our case. Anyone with valid credentials can generate a presigned URL. However, if the generating user does not have permission to perform the intended operation (for example, `get_object`), then the operation will eventually fail.

## There's more...

We can also presign a URL using an SDK such as the Python SDK. With Python SDK, we also use the `boto3` library for Python. Boto is the AWS SDK for Python. It facilitates the creation, configuration, and management of AWS services, such as EC2 and S3, using Python.

## See also

Refer to the following link for more use cases for Python and Boto3 related to presigned URLs: <https://boto3.amazonaws.com/v1/documentation/api/latest/guide/s3-presigned-urls.html>.

## Protecting files with S3 versioning and object locking

In this recipe, we will learn to implement **S3 Object Locking**. S3 Object Lock is a feature that enables users to prevent the deletion or overwriting of objects in S3 for a specified period of time or indefinitely. This feature is essential for meeting regulatory compliance requirements and for implementing data protection strategies. With Object Lock, you can apply a **Write Once, Read Many (WORM)** model, ensuring that data cannot be altered or deleted until the specified retention period expires. This makes it an effective tool for safeguarding critical business and compliance-sensitive data.

S3 versioning is a prerequisite for S3 Object Locking. Hence, we will also quickly look into how to enable versioning from within the *Getting ready* section. If versioning is enabled for a bucket, S3 keeps a copy of every version of the file within the bucket. Versioning thus protects data by providing a means to recover it in the case of unintentional actions such as deletes and overwrites, and hence may also be considered a security-related feature.


## Getting ready

We need the following to successfully complete this recipe:

- A working AWS account; I will be using the `awsseccb-sandbox-1` account that we created in *Chapter 1*.
- An S3 bucket with **Bucket Versioning** enabled; I will use a bucket named `seccbbucket` with a file named `image-cloudericks.png`.

We can enable versioning while creating an S3 bucket as follows:

### Bucket Versioning

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. [Learn more](#) 

### Bucket Versioning

- ☐ Disable
- ☒ Enable

Figure 4.16 – Enabling versioning for a new bucket



We can also enable S3 versioning for an existing bucket as follows. Navigate to the S3 bucket's **Properties** tab and click on **Edit** next to **Bucket Versioning**. Under **Bucket Versioning**, select **Enable**, and then click on **Save changes**.

The screenshot shows the 'Edit Bucket Versioning' page in the AWS Management Console. The breadcrumb navigation at the top reads: [Amazon S3](#) > [Buckets](#) > [seccbucket](#) > **Edit Bucket Versioning**. The main heading is 'Edit Bucket Versioning' with an 'Info' link. Below this is a section titled 'Bucket Versioning' with a descriptive paragraph and a 'Learn more' link. The 'Bucket Versioning' section contains two radio button options: 'Suspend' (unselected) and 'Enable' (selected). Below the options is a blue information box stating: 'After enabling Bucket Versioning, you might need to update your lifecycle rules to manage previous versions of objects.' At the bottom of the main content area is the 'Multi-factor authentication (MFA) delete' section, which is currently 'Disabled'. At the bottom right of the console window are two buttons: 'Cancel' and 'Save changes'.

[Amazon S3](#) > [Buckets](#) > [seccbucket](#) > **Edit Bucket Versioning**

## Edit Bucket Versioning [Info](#)

**Bucket Versioning**

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. [Learn more](#)

**Bucket Versioning**

☐ Suspend  
This suspends the creation of object versions for all operations but preserves any existing object versions.

☒ **Enable**

**i** After enabling Bucket Versioning, you might need to update your lifecycle rules to manage previous versions of objects.

**Multi-factor authentication (MFA) delete**

An additional layer of security that requires multi-factor authentication for changing Bucket Versioning settings and permanently deleting object versions. To modify MFA delete settings, use the AWS CLI, AWS SDK, or the Amazon S3 REST API. [Learn more](#)

Disabled

**Cancel** **Save changes**

Figure 4.17 – Enabling versioning for an existing bucket

Next, we will enable Object Lock for our bucket.

## How to do it...

We can enable Object Locking as follows:

1. Login into the AWS Management Console and navigate to the **S3** service.
2. Navigate to the S3 bucket, go to the bucket's **Properties** tab, and click on **Edit** next to **Object Lock**.
3. On the **Edit Object Lock** screen, select **Enable** and also select the checkbox to acknowledge that enabling Object Lock will permanently allow objects in this bucket to be locked.

### Edit Object Lock [Info](#)

**Object Lock**

Store objects using a write-once-read-many (WORM) model to help you prevent objects from being deleted or overwritten for a fixed amount of time or indefinitely. Object Lock works only in versioned buckets. [Learn more](#)

☐ Disable

☒ **Enable**

Permanently allows objects in this bucket to be locked. Additional Object Lock configuration is required in bucket details after bucket creation to protect objects in this bucket from being deleted or overwritten.

**Enabling Object Lock will permanently allow objects in this bucket to be locked**

After you enable Object Lock for a bucket, you can't disable Object Lock or suspend Versioning for that bucket. Learn more about [Using Object Lock](#)

☒ I acknowledge that enabling Object Lock will permanently allow objects in this bucket to be locked.

**Default retention**

Automatically protect new objects put into this bucket from being deleted or overwritten.

☒ **Disable**

☐ Enable

[Cancel](#) [Save changes](#)

Figure 4.18 – Enabling Object Lock

4. Enable default retention to protect new objects put into the bucket by selecting **Enable** under **Default retention**. Once we enable default retention, we should get additional options to configure Object Locking.

**Default retention**  
Automatically protect new objects put into this bucket from being deleted or overwritten.

☐ Disable  
☒ **Enable**

**Default retention mode**

☒ **Governance**  
Users with specific IAM permissions can overwrite or delete protected object versions during the retention period.

☐ **Compliance**  
No users can overwrite or delete protected object versions during the retention period.

**Default retention period**


Must be a positive whole number.

**Cancel** **Save changes**

Figure 4.19 – Enabling default retention

5. Select **Governance** under **Default retention mode**, enter 30 for **Default retention period**, and click on **Save changes**.
6. Upload a new object to our S3 bucket. I have uploaded an `image-heartin.png` file.
7. Go to the bucket's **Object** tab and click on the new object's name to go to the object's **Properties** tab.
8. Scroll down and verify the retention configuration under **Object Lock retention**.

## Object Lock retention

Store objects using a write-once-read-many (WORM) model to help you prevent objects from being deleted or overwritten for a fixed amount of time or indefinitely. Object Lock works only in versioned buckets. [Learn more](#) 

Edit

Retention mode

Governance mode

Retain until date

January 2, 2024, 08:57:04 (UTC+05:30)

Figure 4.20 – Object Lock retention for new objects after setting the default retention mode


9. Optionally, we can click on **Edit** on the **Object Lock retention** screen, as shown in *Figure 4.20*, to do one of the following: **Disable** retention, change **Retention mode** from **Governance mode** to **Compliance mode**, or extend the retention date using the **Retain until date** option.

### Important Note

In governance mode, objects in S3 Object Lock are unchangeable until the set retention period is over. To remove these objects before the retention date, retention can be disabled after clicking on **Edit**, as shown in *Figure 4.20*. In compliance mode, we won't be able to disable retention and objects remain unchangeable until their retention period ends. The only way to delete these objects before the retention time expires is by closing the associated AWS account. In both modes, we can extend the retention date to a future date, but we cannot reduce it.

10. Optionally, go to any existing object's **Properties** tab, scroll down to **Object Lock retention**, and click **Edit** to enable Object Lock for that object. We should get a screen similar to the screen for configuring default retention, as shown in *Figure 4.19*, where we can configure retention for this object.
11. Optionally, go to any new or existing object's **Properties** tab, scroll down to **Object Lock legal hold**, and click on **Edit** to enable **Legal hold**, as shown in the following figure.

### Object Lock legal hold

Prevent objects from being deleted or overwritten until the hold is explicitly removed. Legal hold can be turned on or off by AWS accounts that have specific IAM permissions. [Learn more](#) 

Legal hold

☐ Disable

☒ Enable

Figure 4.21 – Enabling Object Lock legal hold

Optionally, wait for the retention date to expire and then try to delete the objects for which we added Object Lock in compliance mode.

## How it works...

Amazon S3 Object Lock is a feature designed to protect your S3 objects from being deleted or overwritten. It's particularly useful in enforcing data retention policies for compliance, legal, or regulatory requirements. To use Object Lock, you must first enable versioning in your S3 bucket. Versioning is a means by which S3 keeps multiple versions of an object in the same bucket, allowing you to preserve, retrieve, and restore every version of every object stored in your bucket. With versioning enabled, Object Lock can protect individual object versions.

Object Lock has two primary retention modes: Governance and Compliance. In Governance mode, users with specific permissions can overwrite or delete an object version before its retention period expires. This mode is ideal for internal data management, as it provides a balance between protection and flexibility. In contrast, Compliance mode offers a stricter level of protection. Once an object is locked in Compliance mode, no one can overwrite or delete the object version until its retention period has passed, not even the root user in the AWS account. This mode is designed for scenarios where data must be retained to comply with legal or regulatory standards.

Additionally, Object Lock provides a feature called legal hold for an added layer of protection. A legal hold can be applied to objects regardless of their retention mode, and it prevents object version deletion without any time-bound constraints. Legal Holds are ideal for situations where you might need to retain an object for legal reasons. It's important to note that both the retention modes and Legal Hold can be applied to individual objects or as default settings at the bucket level. Together, these features make S3 Object Lock a robust tool for ensuring that critical data is not deleted or altered, thereby helping organizations adhere to compliance requirements and protect their data from accidental loss or deletion.

## There's more...

After learning about S3 Object Lock in this recipe, let us quickly look into **Glacier Vault Lock**, an important AWS feature that helps keep data secure within AWS Glacier, especially when the data

needs to be stored for a long time. At the core of Glacier Vault Lock is the WORM principle. This means that after data is stored, it can be accessed but not altered. By establishing and locking a policy, we can ensure that no one, not even administrators or creators, can modify or erase these rules or the data they protect. These policies can range from basic ones, such as setting a data retention period, to more detailed ones specifying deletion permission roles or even creating a permanent archive.

Setting up a Glacier Vault Lock involves the following steps:

1. **Start a Glacier Vault:** Begin by creating a vault in Amazon Glacier using the AWS Management Console.
2. **Create a Vault Lock policy:** Develop a policy that outlines your data retention preferences. This policy defines the usage and alteration guidelines for the data in the vault.
3. **Initiate the Vault Lock:** Apply your policy to the vault. This begins the process but doesn't activate the policy immediately.
4. **Review and finalize:** AWS provides a specific timeframe (typically 24 hours) for you to review and finalize your policy. No changes are allowed after this period.
5. **Lock the policy:** Conclude this waiting period by confirming and permanently locking your policy. Once locked, the policy is irreversible, ensuring the data's safety according to your established rules.

By following these steps, Glacier Vault Lock can effectively be utilized to safeguard critical data requiring long-term storage, meeting compliance and security requirements.

Other important security features related to S3 include the following:

- **S3 encryption:** Amazon S3 encryption provides methods to protect our data at rest. There are two main types: SSE, whereby Amazon S3 encrypts our data as it writes it to disks and decrypts it when we access it, and **client-side encryption**, whereby we encrypt data on our side before uploading it to S3. For SSE, we have options such as **S3-Managed Keys (SSE-S3)**, **AWS Key Management Service Keys (SSE-KMS)**, and **Customer-Provided Keys (SSE-C)**. We will discuss more about S3 Encryption later in this book after we discuss about the **AWS Key Management Service (KMS)**.
- **S3 Access Points:** Amazon S3 Access Points facilitate data access management for large-scale, shared data sets in S3 by providing unique hostnames with tailored access policies. For instance, if we are operating a data lake with multiple departments accessing the same S3 bucket, we can create separate access points for each department – such as `finance-data-access` or `hr-data-access` – each with its specific permissions and network controls. This allows us to customize access according to each department's needs, ensuring secure and efficient data handling. S3 Access Points thus offer a streamlined solution for scenarios requiring distinct, role-based access controls, significantly simplifying the management of complex data access requirements.

- **S3 access logs:** Amazon S3 access logs are a crucial component for enhancing security and monitoring activity in our S3 buckets. These logs provide detailed records of all requests made to a specific S3 bucket, including requester information, bucket name, request time, and the action performed. This information is vital for security analysis and audit trails, allowing us to track access patterns, identify suspicious activities, and ensure compliance with security policies. By analyzing these logs, we can gain insights into usage trends and potential security vulnerabilities, as well as improve our overall data protection strategy in S3.
- **S3 replication:** Amazon S3 replication, encompassing both **Cross-Region Replication (CRR)** and **cross-account replication**, plays a vital role in enhancing data security and availability. With CRR, data is automatically replicated across multiple AWS regions, providing geographical diversification, which is crucial for disaster recovery and meeting data residency requirements. Cross-account replication, meanwhile, offers an added security layer by replicating data between buckets in different AWS accounts. This is ideal for scenarios such as auditing where data integrity and separation are critical. Both replication strategies can be governed by specific policies, allowing precise control over which data gets replicated and the manner of its replication.

## See also

- Check the *Using key policies with conditional keys* recipe in *Chapter 3* to see how an AWS KMS key can be used for encryption in an S3 bucket.
- While this book addresses key S3 security features, providing recipes for every single one is beyond its scope. For an always up-to-date and complete list, you can visit the blog post at <https://www.cloudericks.com/blog/a-comprehensive-list-of-s3-security-features-within-aws-cloud>.

## Encrypting data on S3

In this recipe, we will learn to encrypt data on S3 at rest using SSE techniques. Encryption on the server side can be done in three ways: SSE-S3, SSE-KMS, and SSE-C. In client-side encryption, data is encrypted on the client side and then sent to the server.

## Getting ready

We need a working AWS account with the following resources configured:

- We need an S3 bucket. I will be using a bucket with the `awssecbucket` name. Replace this with your bucket name.
- We need a user with **AdministratorAccess** permission. Configure a CLI profile for this user. I will be calling both the user and the profile on the `awssecadmin` CLI.
- We need a customer-managed key created in KMS. Follow the *Creating keys in KMS* recipe from *Chapter 3* to create a key. I have created one named `MyS3Key`.

## How to do it...

In this recipe, we will learn about various use cases for SSE.

### SSE with SSE-S3

We can upload an object from the console with SSE-S3 as follows:

1. Navigate to your S3 bucket in the console.
2. Under the **Objects** pane, click on **Upload | Add Files**, select your file, and then click on **Upload**.
3. Now go to the **Properties** tab. In our bucket, scroll down to the **Default encryption** pane and click on **Edit**. Under **Encryption type**, select **Server-side encryption with Amazon S3 managed keys (SSE-S3)**, and under **Bucket Key**, select **Enable**. Finally, click on **Save changes**.

**Default encryption**  
Server-side encryption is automatically applied to new objects stored in this bucket.

Encryption type [Info](#)

- ☒ Server-side encryption with Amazon S3 managed keys (SSE-S3)
- ☐ Server-side encryption with AWS Key Management Service keys (SSE-KMS)
- ☐ Dual-layer server-side encryption with AWS Key Management Service keys (DSSE-KMS)  
Secure your objects with two separate layers of encryption. For details on pricing, see [DSSE-KMS pricing](#) on the **Storage** tab of the [Amazon S3 pricing page](#).

**Bucket Key**  
Using an S3 Bucket Key for SSE-KMS reduces encryption costs by lowering calls to AWS KMS. S3 Bucket Keys aren't supported for DSSE-KMS. [Learn more](#)

- ☐ Disable
- ☒ Enable

[Cancel](#) [Save changes](#)

Figure 4.22 – The default encryption options

#### Important note

It is important to note that, if we try to open or download the object, we will still be able to see the object as-is because S3 will decrypt the object using the same key.



## SSE with SSE-KMS

We can upload an object from the console with SSE-KMS as follows:

1. Navigate to your S3 bucket.
2. Now go to the **Properties** pane. In the **Properties** tab, scroll down to the **Default encryption** pane and click on **Edit**. Under **Encryption type**, select **Server-side encryption with AWS Key Management Service keys (SSE-KMS)**; under the **AWS KMS key** pane, select **Choose from your AWS KMS keys** and select the key; under **Bucket Key**, select **Enable**, and finally, click on **Save changes**.

Encryption type [Info](#)

☐ Server-side encryption with Amazon S3 managed keys (SSE-S3)  
☒ Server-side encryption with AWS Key Management Service keys (SSE-KMS)  
☐ Dual-layer server-side encryption with AWS Key Management Service keys (DSSE-KMS)  
 Secure your objects with two separate layers of encryption. For details on pricing, see [DSSE-KMS pricing](#) on the **Storage** tab of the [Amazon S3 pricing page](#).

AWS KMS key [Info](#)

☒ Choose from your AWS KMS keys  
☐ Enter AWS KMS key ARN

Available AWS KMS keys

arn:aws:kms:us-east-1:201882936474:key/9b03e... ▼ [↻](#) [Create a KMS key](#)

Bucket Key

Using an S3 Bucket Key for SSE-KMS reduces encryption costs by lowering calls to AWS KMS. S3 Bucket Keys aren't supported for DSSE-KMS. [Learn more](#)

☐ Disable  
☒ Enable

⚠ Changing the default encryption settings might cause in-progress replication and Batch Replication jobs to fail. These jobs might fail because of missing AWS KMS permissions on the IAM role that's specified in the replication configuration. If you change the default encryption settings, make sure that this IAM role has the necessary AWS KMS permissions. [Learn more](#)

[Cancel](#) [Save changes](#)

Figure 4.23 – Changing the encryption options with KMS

3. We can change encryption for an existing object to SSE-KMS as follows. Go to the object's **Properties** tab. Scroll down to the **Server-side encryption settings** pane and click on **Edit**. Under **Encryption settings**, select **Use bucket settings for default encryption** and click on **Save changes**.

Encryption settings



☒ Use bucket settings for default encryption

☐ Override bucket settings for default encryption

Encryption type [Info](#)


Server-side encryption with AWS Key Management Service keys (SSE-KMS)

Encryption key ARN

 <arn:aws:kms:us-east-1:201882936474:key/9b03e729-a8e0-40b1-a171-9bc2677708f8> 



Bucket Key


When KMS encryption is used to encrypt new objects in this bucket, the bucket key reduces encryption costs by lowering calls to AWS KMS.

[Learn more](#) 

Enabled

**Specified objects**

 1 

Name	Type	Last modified	Size	Storage class
 <a href="#">image-cloudericks.png</a>	png	November 21, 2023, 18:16:11 (UTC+05:30)	7.3 KB	Standard

Cancel

Save changes

Figure 4.24 – Changing the encryption options for an existing object

4. We can upload an object from the CLI with SSE-KMS using the following command:

```
aws s3 cp image-heartin-k.png s3://awsseccookbook/image-  
heartin-k.png \  
--sse aws:kms \  
--sse-kms-key-id cd6b3dff-cfe1-45c2-b4f8-b3555d5086df \  
--profile awssecadmin
```

We will get the following response:

```
C:\Users\sivai>aws s3 cp C:\Users\sivai\OneDrive\Pictures\image-cloudericks.  
png s3://awssecbucket/C:\Users\sivai\OneDrive\Pictures\image-cloudericks.png  
--sse aws:kms --sse-kms-key-id 9b03e729-a8e0-40b1-a171-9bc2677708f8 --profi  
le mikey098  
upload: OneDrive\Pictures\image-cloudericks.png to s3://awssecbucket/C:\User  
s\sivai\OneDrive\Pictures\image-cloudericks.png
```

Figure 4.25 – The response for uploading an object from the CLI with SSE-KMS

**Important Note**

`sse-kms-key-id` is the ID of the KMS key you created (refer to the *Getting ready* section for more details).

**SSE with SSE-C**

We can upload an object from the CLI with SSE-C as follows:

1. Upload an object from the CLI with SSE-C by using the following command. Provide any set of digits as the key. You will need to use the same to retrieve the object later:

```
aws s3 cp image-heartin-k.png s3://awsseccookbook/
imageheartin-k.png \
--sse-c AES256 \
--sse-c-key 12345678901234567890123456789012 \
--profile awssecadmin
```

We will get the following response:

```
C:\Users\sivai>aws s3 cp C:\Users\sivai\OneDrive\Pictures\image-cloudericks.
png s3://awsseccbucket/C:\Users\sivai\OneDrive\Pictures\image-cloudericks.png
--sse-c AES256 --sse-c-key 12345678901234567890123456789012 --profile mikey
098
upload: OneDrive\Pictures\image-cloudericks.png to s3://awsseccbucket/C:\User
s\sivai\OneDrive\Pictures\image-cloudericks.png
```

Figure 4.26 – The response for uploading an object from the CLI with SSE-C

2. Retrieve the object encrypted using SSE-C, providing the same key we used in the previous command, as follows:

```
aws s3 cp s3://awsseccookbook/image-heartin-k.png
imageheartin-k1.png \
--sse-c AES256 \
--sse-c-key 12345678901234567890123456789012 \
--profile awssecadmin
```

We will get the following response:

```
C:\Users\sivai>aws s3 cp s3://awsseccbucket/C:\Users\sivai\OneDrive\Pictures\
image-cloudericks.png C:\Users\sivai\OneDrive\Pictures\image-cloudericks.png
--sse-c AES256 --sse-c-key 12345678901234567890123456789012 --profile mikey
098
download: s3://awsseccbucket/C:\Users\sivai\OneDrive\Pictures\image-clouderic
ks.png to OneDrive\Pictures\image-cloudericks.png
```

Figure 4.27 – The response for retrieving an object encrypted using SSE-C

**Tip**

If we do not specify the `sse-c` option while downloading an object encrypted with SSE-C, we will get a fatal error: An error occurred (400) exception when calling HeadObject operation: Bad Request. If we do not specify the correct key that was used for encryption (using the `sse-c-key` option) while downloading an object encrypted with SSE-C, we will get a fatal error: An error occurred (403) exception when calling HeadObject operation: Forbidden.

**How it works...**

In the *SSE with SSE-S3* section, we uploaded an object from the console with SSE-S3 encryption. We changed the encryption for an existing object to SSE-S3 encryption. We also uploaded an object with SSE-S3 encryption. When performing SSE-S3 encryption from the CLI, the value of the `sse` parameter is optional. The default is AES256.

In the *SSE with SSE-KMS* section, we uploaded an object with SSE-KMS encryption. We changed encryption for an existing object to SSE-KMS encryption. We also uploaded an object from the CLI with SSE-KMS encryption. When performing SSE-KMS encryption from the CLI, the value of the `sse-c` parameter is optional. The default is AES256.

In the *SSE with SSE-C* section, we uploaded an object from the CLI with SSE-C encryption. Unlike the other two SSE techniques, SSE-S3 and SSE-KMS, the console does not currently have an explicit option for SSE-C. We will need to execute this using APIs. In this recipe, we used a 32-digit number as the key. However, in the real world, keys are generally generated using a key generation tool. We will learn more about keys when we discuss KMS later in this book.

**There's more...**

Let's quickly go through some important concepts related to S3 encryption:

- Data on S3 can be encrypted while at rest (stored on AWS disks) or in transit (moving to and from S3). Encryption at rest can be done using SSE or by uploading encrypted data from the client.
- S3 SSE techniques for data at rest use symmetric keys for encryption.
- Encryption of data in transit using SSL/TLS (HTTPS) uses asymmetric keys for encryption.
- S3 default encryption (available as bucket properties) provides a way to set the default encryption behavior for an S3 bucket with SSE-S3 or SSE-KMS. Enabling this property does not affect existing objects in our bucket and applies only to new objects uploaded.
- With client-side encryption, we need to manage keys on our own. We can also use KMS to manage keys through SDKs. However, it is not currently supported by all SDKs.

- Encryption in transit can be achieved with client-side encryption or by using SSL/TLS (HTTPS).
- SSE types, SSE-S3 and SSE-KMS, follow envelope encryption, while SSE-C does not use envelope encryption.

Some important features of SSE-S3 include the following:

- AWS takes care of all key management.
- It follows envelope encryption.
- It uses symmetric keys to encrypt data.
- Each object is encrypted with a unique key.
- It uses the AES-256 algorithm.
- A data key is encrypted with a master key that is automatically rotated periodically.
- It is free.

Some important features of SSE-KMS include the following:

- Keys are managed by AWS KMS.
- Keys can be shared by multiple services (including S3).
- As customers, we get more control over keys, such as creating master and data keys, and disabling and rotating master keys.
- It follows envelope encryption.
- It uses symmetric keys to encrypt data.
- A data key is encrypted with a master key.
- It uses the AES-256 algorithm.
- We can choose which object key to encrypt while uploading objects.
- We can use CloudTrail to monitor KMS API calls, enabling better auditing.
- It is not free.

Some important features of SSE-C include the following:

- Keys are managed by us (customers).
- The customer provides a key along with data; S3 uses this key for encryption and deletes the key.
- The key must be supplied for decryption as well.
- It does not use envelope encryption.
- It uses symmetric keys to encrypt data.

- It uses the AES-256 algorithm.
- AWS will force you to use HTTPS while uploading data since you are uploading your symmetric key as well.
- It is free.
- By default, S3 allows both HTTP and HTTPS access to data; HTTPS can be enforced with the help of a bucket policy with the following condition element:

```
"Condition": {  
  "Bool": {  
    "aws:SecureTransport": "false"  
  }  
}
```

Any requests without HTTPS will fail with this condition.

## See also

- Understand the basics of Amazon S3 encryption from <https://www.cloudericks.com/blog/understanding-amazon-s3-encryption>.
- Understand more about S3 bucket key for SSE-KMS encryption from <https://www.cloudericks.com/blog/understanding-s3-bucket-key-for-sse-kms-encryption>.
- Read more about encryption at <https://www.secdops.com/blog/getting-started-with-encryption>.



# 5

## Network and EC2 Security with VPCs

An Amazon **Virtual Private Cloud (VPC)** is a foundational component within the AWS cloud that allows the creation of private networks within the AWS cloud, distinct and isolated from the broader AWS public cloud. It enables us to launch AWS resources into a **Virtual Private Network (VPN)** that we have defined within our AWS account. This virtual network is similar to a traditional network we might run in our own data center but comes with the added advantage of AWS's scalable infrastructure.

Users have complete control over their virtual network environment. This means they can choose their IP address ranges, set up public and private subnets, and customize route tables and network gateways to suit their requirements. This flexibility enables the deployment of internet-accessible instances, such as web servers, in public subnets, while positioning internal-use instances, such as database servers, in private subnets.

Additionally, Amazon VPC enhances security with a robust suite of features such as **security groups**, **Network Access Control Lists (NACLs)**, **flow logs**, VPN connections, integration with AWS **Identity and Access Management (IAM)**, **PrivateLink**, **endpoint services**, and **gateway endpoints**. Together, these security features significantly fortify the security measures within a VPC, creating a secure, robust, and controlled network environment for the deployment and management of AWS resources.

This chapter includes the following recipes:

- Setting up VPC plus VPC resources with minimal effort
- Creating a bare VPC and setting up public and private subnets
- Launching an EC2 instance with a web server using user data
- Creating and configuring security groups



- Working with NACLs
- Using a VPC gateway endpoint to connect to S3
- Configuring and using VPC flow logs
- Setting up and configuring NAT gateways

## Technical requirements

Before diving into the recipes of this chapter, we need to ensure we have the following knowledge and requirements in place:

- We need an active AWS account to complete the recipes within this chapter. We can use an account that is part of an AWS Organization or a standalone account. I will be using the `awssecb-sandbox-1` account that we created in the *Multi-account management with AWS Organizations* recipe in *Chapter 1*. However, I won't be utilizing any AWS Organizations features, meaning you can follow these steps with a standalone account too.
- For administrative actions, we need a user who has **AdministratorAccess** permission to the AWS account we are working with. This can be an IAM Identity Center user or an IAM user. I will be using the `awssecbadmin1` IAM Identity Center user we created in the *User management and SSO with IAM Identity Center* recipe in *Chapter 1*. However, I won't be utilizing any IAM Identity Center features, meaning you can follow these steps with an IAM user, too, if the user has **AdministratorAccess** permission within the account. You can create an IAM user by following the *Setting up IAM, account aliases, and billing alerts* recipe in *Chapter 1*.
- It is beneficial to have a foundational understanding of essential computer networking concepts. You can learn the computer networking concepts needed for this chapter at <https://www.secdops.com/blog/essential-computer-networking-concepts-for-the-cloud>.

The code files for this book are available at <https://github.com/PacktPublishing/AWS-Security-Cookbook-Second-Edition>. The code files for this chapter are available at <https://github.com/PacktPublishing/AWS-Security-Cookbook-Second-Edition/tree/main/Chapter05>.

## Setting up VPC plus VPC resources with minimal effort

In this recipe, we will create a VPC along with network resources such as **public subnets**, **private subnets**, **route tables**, **Internet Gateway (IGW)**, **Network Address Translation (NAT) gateway**, and **VPC Endpoints (S3 Gateway)**. We can also create these components individually if needed, as we will see in subsequent recipes within this chapter. To explore all possibilities, I will be selecting all network resource components available to select in this recipe. Some of these components, such as the NAT gateway, have associated charges. Select components and their quantity as per your needs.

## Getting ready

To follow this recipe, we need a working `awssecb-sandbox-1` AWS account, and a `awssecbadmin1` user, as described in the *Technical requirements* section.

## How to do it...

Please also note that the default options shown here may change over time, so check the preview to make sure you are only creating the resources that you need before creating the VPC. Let us get started:

1. Log into the AWS Management Console and go to the **VPC** service.
2. From the left sidebar, under the **Virtual private cloud** heading, click on **Your VPCs**. We will be taken to the **Your VPCs** page, where we can see our VPCs. If we are using VPCs for the first time, we will only see the default VPC.
3. On the **Your VPCs** page, click on **Create VPC**. Within the **Create VPC** screen, select **VPC and more**, then select **Auto-generate**. After that, provide the `awssecb` project prefix, as shown in the following figure:

**Create VPC** [Info](#)

A VPC is an isolated portion of the AWS Cloud populated by AWS objects, such as Amazon EC2 instances. Mouse over a resource to highlight the related resources.

**VPC settings**

**Resources to create** [Info](#)  
Create only the VPC resource or the VPC and other networking resources.

☐ VPC only ☒ VPC and more

**Name tag auto-generation** [Info](#)  
Enter a value for the Name tag. This value will be used to auto-generate Name tags for all resources in the VPC.

☒ Auto-generate

awssecb

Figure 5.1 – Creating a VPC with network resources

4. Provide the `10.0.0.0/16` value under **IPv4 CIDR block**.
5. For **IPv6 CIDR block**, we have two options: **No IPv6 CIDR block** and **Amazon-provided IPv6 CIDR block**. Select **No IPv6 CIDR block**.
6. For **Tenancy**, there are two options: **Default** and **Dedicated**. Select **Default**.
7. For **Number of Availability Zones (AZs)**, select 2; for **Number of public subnets**, select 2; and for **Number of private subnets**, select 2, as shown in the following figure:

**Number of Availability Zones (AZs)** [Info](#)

Choose the number of AZs in which to provision subnets. We recommend at least two AZs for high availability.

1	2	3
---	---	---

► **Customize AZs**

---

**Number of public subnets** [Info](#)

The number of public subnets to add to your VPC. Use public subnets for web applications that need to be publicly accessible over the internet.

0	2
---	---

**Number of private subnets** [Info](#)

The number of private subnets to add to your VPC. Use private subnets to secure backend resources that don't need public access.

0	2	4
---	---	---

► **Customize subnets CIDR blocks**

Figure 5.2 – Configuring the number of AZs and subnets

Optionally, we can select AZs for our subnets after expanding **Customize AZs** and customize the CIDR blocks using the **Customize subnets CIDR blocks** option.

8. For **NAT gateways (\$)**, select **1 in 1 AZ**; for **VPC endpoints**, select **S3 Gateway**; and for **DNS options**, select **Enable DNS hostnames** and **Enable DNS resolution**.

**NAT gateways (\$)** [Info](#)  
 Choose the number of Availability Zones (AZs) in which to create NAT gateways.  
 Note that there is a charge for each NAT gateway

**None** **In 1 AZ** **1 per AZ**

**VPC endpoints** [Info](#)  
 Endpoints can help reduce NAT gateway charges and improve security by accessing S3 directly from the VPC. By default, full access policy is used. You can customize this policy at any time.

**None** **S3 Gateway**

**DNS options** [Info](#)  
☒ Enable DNS hostnames  
☒ Enable DNS resolution

► **Additional tags**

**Cancel** **Create VPC**

Figure 5.3 – Configuring NAT gateways, VPC endpoints and DNS options

AWS will show us a preview of the VPC in the right pane of the page and network resources it will create as follows:

- A VPC with the `awssecceb-vpc` name
- Subnets named `awssecceb-subnet-public1-us-east-1a`, `awssecceb-subnet-private1-us-east-1a`, `us-east-1b`, `awssecceb-subnet-public2-us-east-1b`, and `awssecceb-subnet-private2-us-east-1b`
- Route tables named `awssecceb-rtb-public`, `awssecceb-rtb-private1-us-east-1a`, and `awssecceb-rtb-private2-us-east-1b`
- Network connections named `awssecceb-igw`, `awssecceb-nat-public1-us-east-1a` and `awssecceb-vpce-s3`

### Important note

A single route table is shared among all public subnets while there is a route table per private subnet. Even though this is not shown in the preview, a main route table is also created that dictates the default rules for subnets not associated with any route table. The selection of public subnets necessitates the creation of an IGW, which is then linked to these public subnets, facilitating internet access. Additionally, NAT gateways, which provide internet access to instances in private subnets while preventing direct inbound internet traffic, must be deployed within a public subnet and, thus, also rely on the existence of an IGW for outbound internet traffic.

9. Click on **Create VPC** to create the VPC with the network resources shown in the preview.
10. Verify that all the resources that were listed in the preview have been created.

In this recipe, we created a VPC along with VPC resources. To create only a VPC without additional resources, we can select the **VPC only** option instead of **VPC and more** as we saw in *Figure 5.1*.

## How it works...

In this recipe, we successfully configured a VPC and its associated network resources. Let us learn more about these components. A VPC is a virtual network dedicated to our AWS account. It is isolated from other virtual networks in the AWS Cloud. It allows us to launch AWS resources, such as EC2 instances, into a network that we have defined.

The **IPv4 CIDR block** defines a range of private IPv4 addresses allocated to our VPC, facilitating internal communication within our VPC while keeping it isolated from external networks. In this recipe, we chose a `10.0.0.0/16` **Classless Inter-Domain Routing (CIDR)** block for our VPC, representing the broadest range of addresses AWS permits for a VPC. CIDR is a method for allocating IP addresses and routing IP packets. It allows multiple IP addresses to be represented as a single expression, significantly simplifying network configuration and management. CIDR is a slightly complex topic for beginners. You can learn more about CIDR at <https://www.secdops.com/blog/understanding-ip-addresses-and-subnetting>.

**IPv6 CIDR Block** is an optional range of IPv6 addresses for our VPC. Selecting **No IPv6 CIDR block** means our VPC will only use IPv4 addresses. IPv6 addresses provide a much larger address space and are becoming increasingly important for future-proofing and global reach.

The **Tenancy** option lets us choose between a shared or dedicated hardware host for our VPC. The **Default** option allows AWS to place our instances on any shared hardware, which is sufficient for most use cases. The **Dedicated** option is typically used for compliance or regulatory requirements.

**Availability Zones (AZs)** are distinct locations within a region engineered to be isolated from failures in other AZs. By selecting multiple AZs, we can achieve high availability. Each subnet is tied to a specific AZ for fault tolerance and low latency. In this recipe, we created two AZs.

A subnet is a logical subdivision of an IP network. Within cloud environments such as AWS, subnets facilitate the organization of a VPC by segmenting the network and directing traffic flow between resources. This segmentation allows for the creation of **public subnets**, which provide direct internet access for resources such as web servers, and **private subnets**. These are designed for resources such as database servers that require restricted access, typically only from within the network or a specific subnet, such as a web server subnet.

With the **VPC and more** option when creating a VPC, subnets are allocated equally across AZs. Therefore, we can select subnets in multiples of the number of AZs we selected. If we select **1 AZ**, we get the **0** and **1** options for the public subnet and **0**, **1**, and **2** for the private subnet. Similarly, if we select 3 AZs, we get the **0** and **2** options for the public subnets and **0**, **3**, and **6** for the private subnets. In this recipe, we created four subnets – two public subnets and two private subnets.

An IGW is a resource that allows communication between instances in our VPC and the internet. It is necessary for any subnet that requires direct access to the internet. As we opted to create a public subnet within our recipe, AWS created an IGW and associated our public subnets with the IGW. For public subnets, we also need to route all internet-bound traffic to the IGW. This is done using **route tables**.

Route tables contain a set of rules, known as routes, that determine where network traffic from our subnets or the VPC router is directed. The following are the routes created by AWS in the route table associated with our public subnets:

**rtb-05061d49576a65b8c / awssecb-rtb-public**

< Details **Routes** Subnet associations Edge associations >

**Routes (2)** Both ▼ Edit routes

🔍 Filter routes

< 1 > ⚙️

Destination ▼	Target ▼
0.0.0.0/0	<a href="#">igw-04adce823dc7b8fd6</a>
10.0.0.0/16	local

Figure 5.4 – A route table with a route for an IGW

When we select the **VPC and more** option, a single route table is shared among all public subnets while each private subnet is associated with a distinct route table. Apart from these, there is also a **main route table** created per VPC irrespective of whether we use the **VPC only** or **VPC and more** options. The main route table dictates the default rules for subnets not associated with any route table. By default, no subnets are associated explicitly with the main route table.

All subnets not associated with any route table explicitly within a VPC are associated implicitly with the main route table. This means that the main route table's routing behavior applies to all subnets not associated with any other route tables. In this recipe, all subnets are associated with a route table and hence there will not be any association for the main route table. We can experiment by removing a subnet's association with a route table and we can see that it will be implicitly associated with the default route table. We can set a different route table as the main route table if we want to.

Consequently, with a configuration of two public subnets and two private subnets, there will be four route tables: one for the public subnets, one for each private subnets, and one main route table. If the configuration includes two public subnets and four private subnets, the total rises to six route tables: one for the public subnets, one for each of the private subnets, and one main route table.

A NAT gateway allows instances in a private subnet to initiate outbound internet traffic but prevents unsolicited inbound traffic from the internet. This is crucial for updating, patching, or downloading dependencies for instances that do not need to be directly accessible from the outside world. NAT is a method that enables private subnet resources to access the internet or other network services without exposing their private IP addresses. By translating these private IP addresses to a public one, NAT gateways facilitate secure internet access for services within a private subnet, ensuring that the internal structure of a private network remains shielded from external traffic and threats. NAT gateways are deployed on a public subnet and therefore, if we opt not to create any public subnet with the **VPC and more** option and try to select the NAT gateway option, then we will get an error message.

VPC endpoints enable private connections between our VPC and AWS services without requiring traffic to traverse the internet. Selecting an **S3 Gateway** endpoint allows our instances to securely access S3 buckets without an IGW or NAT gateway, improving security and potentially reducing costs. Finally, enabling DNS options such as **DNS hostnames** and **DNS resolution** within our VPC allows our AWS resources to communicate with each other using hostnames instead of IP addresses, simplifying network management and configuration.

Even though it was not shown in the preview while creating the VPC, a NACL was also created. An NACL acts as a firewall for controlling traffic at the subnet level. NACLs examine and filter traffic entering and leaving every subnet within our VPC. When we create the VPC using the **VPC and more** option or the **VPC only** option, a default NACL is created that allows all inbound and outbound traffic. Subnets created are associated automatically with the default NACL. However, we can change the associate to a different NACL. The following are the inbound rules for the default NACL:

Inbound rules (2)

Ed

Filter inbound rules

Rule number ▾	Type ▾	Protocol ▾	Port ra... ▾	Source ▾	Allow/Deny
100	All traffic	All	All	0.0.0.0/0	✔ Allow
*	All traffic	All	All	0.0.0.0/0	✘ Deny

Figure 5.5 – Inbound rules for default NACL

When we created the VPC, a security group called **default** was created by AWS within our VPC. In AWS, a security group functions as a virtual firewall that governs inbound and outbound traffic for EC2 instances. It operates at the instance level, whereas NACLs control traffic at the subnet level. Security groups and NACLs complement each other, providing layered security within AWS environments. We can use security groups for fine-grained, stateful control over individual instances, while NACLs provide an additional layer of stateless, subnet-level security, allowing or blocking types of traffic before they reach your instances.

## There's more...

A default VPC is provided in every AWS region by AWS and includes a public subnet in each AZ, an IGW, and a configured DNS resolution. The default VPC allows the instant launch of Amazon EC2 instances without creating a VPC. The following are some of the important settings of the default VPC:

- Subnets in the default VPC have an outbound route to the internet through an IGW.
- A subnet is created per AZ.
- **DHCP options set** is updated. My default VPC has the following options set: `domain-name = ec2.internal` and `domain-name-servers = AmazonProvidedDNS`.

Let's quickly go through some more important concepts related to AWS VPCs:

- AWS VPCs consist of subresources such as IGWs, route tables, NACLs, subnets, and security groups.
- AWS creates a default VPC ready for us to use in every region. The following are some of its important characteristics:
  - Subnets in the default VPC are routed to the internet.
  - A subnet is created per AZ.
  - **DHCP Options Set** is updated.
- VPC peering can be used to connect one VPC to another through a direct route using private IP addresses, making the associated instances behave as though they are on the same network.
- VPC peering can be done within the same region, across regions, and even across AWS accounts.
- Transitive peering is currently not supported for AWS VPCs. Every VPC must be peered to every other required VPC in a star topology-like structure.
- To avoid the overhead of managing many point-to-point connections with VPC peering, we can make use of the AWS Transit Gateway to connect all the VPCs and even on-premises networks to a single gateway.
- Apart from the standard reserved IP addresses of network address and broadcast address, AWS also reserves three more IP addresses. So, a total of five addresses are reserved in a VPC.

Let's quickly go through some important concepts related to subnets in AWS:

- The first IP address of a subnet represents the subnet ID, while the last IP address represents the subnet's directed broadcast address. Therefore, we cannot use the first and last IP address of a subnet for hosts. AWS reserves additional IP addresses.
- The first IP address of the first subnet of a network represents the subnet ID, as well as the network's ID. Similarly, the last IP address of the last subnet of a network represents the subnet's and the network's directed broadcast address. When using these IP addresses from outside the network, they will be considered as the networks' IPs, and when using them within the network, they will be considered as the subnets' IPs.



- A subnet in AWS VPC is always associated with one AZ. While we cannot have one subnet associated with more than one AZ at a time, we can have multiple subnets associated with a single AZ.
- AWS allows us to choose a subnet without contiguous IP addresses, as shown in the following figure. However, it is good practice to use contiguous IP address ranges, just like we did in this recipe with 10.0.1.0/24 and 10.0.2.0/24.

Name	Subnet ID	State	VPC	IPv4 CIDR
10.0.1.0/24	subnet-015748cfd37cfa5	available	vpc-0b0b918cc87f4da9c   ...	10.0.1.0/24
10.0.2.0/24	subnet-071b1776b379a5e58	available	vpc-0b0b918cc87f4da9c   ...	10.0.2.0/24
10.0.5.0/24	subnet-02fad50229b9f62d4	available	vpc-0b0b918cc87f4da9c   ...	10.0.5.0/24

Figure 5.6 – Subnets without continuous IP addresses

We will learn about network ACLs, security groups, and IGW in later recipes within this chapter.

## See also

- We can read more about default VPC at <https://www.cloudericks.com/blog/understanding-aws-default-vpc>.
- Read more about the main route table at <https://www.cloudericks.com/blog/understanding-the-main-route-table-in-aws-vpc>.
- While we have expanded our understanding of VPCs and their associated network resources, it is important to keep in mind that the primary focus of this book is on security rather than networking. A solid grasp of networking principles is undoubtedly crucial for developing effective security measures; however, delving too deeply into the intricacies of networking could shift our attention away from the broad spectrum of security topics we aim to explore. Readers keen on deepening their networking knowledge can do so at <https://www.cloudericks.com/blog/beginners-roadmap-to-mastering-aws-networking>.

## Creating a bare VPC and setting up public and private subnets

A public subnet is specifically designed to enable instances within it to be accessible from the internet. This is achieved by routing the internet traffic through an IGW by configuring route tables. In the *Setting up VPC plus VPC resources with minimal effort* recipe from this chapter, we selected the **VPC and more** option while creating the VPC as we saw in *Figure 5.1*, which automatically set up public and private subnets, along with a pre-configured IGW and route tables. For this recipe, we will choose the **VPC only** option to create a VPC without the additional networking resources and set up an IGW and a route table to enable internet access to the instances in our subnet.

## Getting ready

To follow this recipe, we need a working `awssecb-sandbox-1` AWS account, and a `awssecbadmin1` user, as described in the *Technical requirements* section.

## How to do it...

First, we will create a bare VPC and a subnet, and then we will set up IGW and route tables.

### Creating a bare VPC

We can create a VPC only without additional resources as follows:

1. Log in to the AWS Management Console and go to the **VPC** service in the console.
2. From the left sidebar, under the **Virtual private cloud** heading, click on **Your VPCs**. We will be taken to the **Your VPCs** page, where we can see our VPCs. If we are using VPCs for the first time, we should see the default VPC.
3. On the **Your VPCs** page, click on **Create VPC**, and on the **Create VPC** screen, select the **VPC only** option. Give the `awssecb-vpc2` name. For **IPv4 CIDR block**, select **IPv4 CIDR manual input** and give the value as `10.0.0.0/24`. For **IPv6 CIDR block**, select **No IPv6 CIDR block**.

**VPC settings**

Resources to create [Info](#)  
Create only the VPC resource or the VPC and other networking resources.

☒ VPC only ☐ VPC and more

Name tag - *optional*  
Creates a tag with a key of 'Name' and a value that you specify.

awssecb-bare-vpc

IPv4 CIDR block [Info](#)  
☒ IPv4 CIDR manual input  
☐ IPAM-allocated IPv4 CIDR block

IPv4 CIDR  
10.0.0.0/24  
CIDR block size must be between /16 and /28.

IPv6 CIDR block [Info](#)  
☒ No IPv6 CIDR block  
☐ IPAM-allocated IPv6 CIDR block  
☐ Amazon-provided IPv6 CIDR block  
☐ IPv6 CIDR owned by me

Tenancy [Info](#)  
Default

Figure 5.7 – Only creating a VPC

4. Scroll down, leave the automatically generated tag (**Key** is Name and **Value** is `awssecceb-vpc2`) as-is, optionally add any new tags, and click **Create VPC**.

Our VPC is created now. Next, we will create a subnet within the VPC.

### ***Creating a subnet within a VPC***

We created a VPC with a CIDR block range of `10.0.0.0/24` in the previous section. We will add a subnet with a netmask of `/25`. We need to create subnets within a VPC's IP address range without any overlap with other subnets. Let us get started.

1. Go to the **VPC** service in the console.
2. Click on **Subnets** in the left sidebar.
3. Click on **Create subnet**.
4. On the **Create subnet** page, under **VPC ID**, select the VPC we created in the previous section using the drop-down box.
5. Under **Subnet settings**, set the **Subnet name** field to `awssecceb-vpc2-public-subnet`. If we plan to create a private subnet, give the `awssecceb-vpc2-private-subnet` name. Complete the remaining steps within the current section, but skip the remaining sections of this chapter.
6. Select **No preference** under **Availability Zone**.
7. For the **IPv4 VPC CIDR block**, keep the default value.
8. For **IPv4 subnet CIDR block**, provide an IP address range that is a subset of our VPC's IP address range. I will be using `10.0.0.0/25`.
9. Scroll down and click on **Create subnet** to create the subnets.
10. Go to the **Subnets** page from the sidebar. We should be able to see the new subnet.

We have created a subnet within this recipe without any internet connectivity yet. If you are looking to create a private subnet, you have done it. To set up the subnet as a public subnet, continue with the rest of the sections of this recipe.

### ***Enable auto-assign public IPv4 address***

To make the subnet a public subnet, it is good to enable the auto-assign public IPv4 address feature as follows:

1. Select the subnet we created, click on the **Actions** dropdown, and click on **Edit subnet settings**.
2. Select **Enable auto-assign public IPv4 address** and click **Save**.
3. Go to the **Subnets** page from the sidebar. We should be able to see **Auto-assign public IPv4 address** set to **Yes** for our first subnet.

Next, we will create an IGW.

## *Creating and configuring an IGW*

We can create and attach an IGW to our VPC for setting up a public subnet as follows:

1. Go to the **VPC** service in the console.
2. Click on **Internet gateways** from the left sidebar.
3. Click on **Create internet gateway**.
4. Give a descriptive name for **Name tag** such as `awsseccb-vpc2-igw`. Leave the automatically generated tag (**Key** is `Name` and **Value** is `awsseccb-vpc2-igw`) as-is, optionally add any new tags, and click **Create internet gateway**. We should get a success message stating that the IGW has been created. If we go to the **Internet gateways** screen, we will see that the **State** of our IGW is currently **Detached**.
5. Select the IGW we created, click on the **Actions** dropdown, and click on **Attach to VPC**.
6. On the **Attach to VPC** screen, select the VPC we created in this recipe and click on **Attach internet gateway**. If we go to the **Internet gateways** screen, we will see that the **State** of our IGW is now **Attached**.

Next, we will create and configure a route table.

## *Creating and configuring a route table*

We can create and configure the route table for setting up a public subnet as follows:

1. Click on **Route tables** from the left sidebar.
2. Click on **Create route table**.
3. Provide the `awsseccb-vpc2-rtb-public` name and select the VPC we created in this recipe.
4. Leave the automatically generated tag (**Key** is `Name` and **Value** is `awsseccb-vpc2-rtb-public`) as-is, optionally add any new tags, and click on **Create route table**.
5. Click on **Route tables** from the left sidebar, select the route table we created, click on the **Actions** dropdown or go to the **Routes** tab, and click on **Edit routes**.
6. On the **Edit routes** page, click **Add route**.
7. For **Destination**, select `0.0.0.0/0`. For **Target**, select **Internet Gateway**, then select the IGW we created in this recipe, and click **Save changes**. If we want to add a route for the IPv6 address, we can add a similar entry with the destination set to `:::/0`.
8. Go to our route table and go to the **Subnet associations** tab.
9. Click on **Edit subnet associations**, select the subnet we created in this recipe, and click on **Save associations**.

We can now launch EC2 instances into our public subnet with the appropriate security group rules and verify the changes.

## How it works...

We first created a bare VPC. When we create a bare VPC using the **VPC only** option, a main route table, a default NACL, and a default security group are also created along with the VPC. The main route table dictates the default rules for subnets not associated with any route table. In our case, the main route table includes only a local route that enables communication within the VPC. This is crucial for allowing instances within the same VPC to communicate with each other without needing to traverse the internet or any other external network.

A network ACL acts as a firewall for controlling traffic at the subnet level. In our case, a default NACL was created that allows all traffic. A security group in AWS acts like a virtual firewall for our EC2 instances to control inbound and outbound traffic. In our case, a default security group was created that will allow all traffic. We will see NACLs and security groups again in later recipes within this chapter.

We selected the **Enable auto-assign public IPv4 address** option, which makes it the default option while creating an EC2 instance within this subnet. We can override this if we want during the instance creation. We can also create and attach an Elastic IP address later to our EC2 instance.

We created and attached an IGW. We also created and configured a route table for internet access. We can make the main route table public if we want to, by editing the routes. However, if we make the main route table public, it will implicitly make all the new subnets public until we associate it with a private route table. Therefore, it is a good practice to create a separate route table for public access and then attach the subnets that need public access to that VPC. We will do this next.

## There's more...

In the previous recipe from this chapter, *Setting up VPC plus VPC resources with minimal effort*, we discussed VPC and VPC resources in detail. Therefore, I will not repeat. Please refer to the *How it works*, *There's more...*, and *See also* sections of that recipe even if you do not want to practice the *How to do it* section.

## See also

- Read more about IGWs at <https://www.cloudericks.com/blog/understanding-internet-gateway-in-aws>.
- Read more about AWS VPC route tables at <https://www.cloudericks.com/blog/understanding-aws-vpc-route-tables>.

## Launching an EC2 instance with a web server using user data

In this recipe, we will use the EC2 user data feature to set up a simple web server during the launch of an EC2 instance. We will use this instance to test the VPC and public subnet we created in the previous recipes. We will also use this recipe in future recipes where we need to launch an EC2 instance. The EC2 user data feature also significantly enhances security by enabling automated security patching, ensuring instances are updated with the latest protections upon booting and reducing vulnerabilities. Moreover, it guarantees consistent security configurations across all instances, fostering a uniform security posture that prevents configuration drifts and strictly adheres to established security standards from the start.

### Getting ready

We need the following to successfully complete this recipe:

- A working `awsseccb-sandbox-1` AWS account, and a `awsseccbadmin1` user, as described in the *Technical requirements* section.
- An `awsseccb-vpc` VPC following the *Setting up VPC plus VPC resources with minimal effort* recipe from this chapter.

### How to do it...

We will first set up an EC2 instance as a web server using EC2 user data and verify it from the browser. Then we will see how we can log into the instance using **Secure Shell (SSH)**.

#### *Setting up a web server using EC2 user data*

We can launch an EC2 instance with a web server using user data as follows:

1. Log into the AWS Management Console and go to the **EC2** dashboard.
2. Click on **Instances** from the left sidebar.
3. Click on **Launch instances** at the top right of the page.
4. For **Name**, provide the `Cloudericks Web Server` value.
5. In the **Application and OS Images (Amazon Machine Image)** section, select **Amazon Linux**, and for **Amazon Machine Image (AMI)**, select **Amazon Linux 2023 AMI**.

▼ Application and OS Images (Amazon Machine Image) [Info](#)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Recents

Quick Start

Amazon Linux

macOS

Ubuntu

Windows

Red Hat

SUSE Linux

Browse more AMIs

Including AMIs from AWS, Marketplace and the Community

Amazon Machine Image (AMI)

Amazon Linux 2023 AMI

ami-051f8a213df8bc089 (64-bit (x86), uefi-preferred) / ami-05adadbbe8cf9fb48 (64-bit (Arm), uefi)

Virtualization: hvm    ENA enabled: true    Root device type: ebs

Free tier eligible ▼

Description

Amazon Linux 2023 AMI 2023.4.20240401.1 x86\_64 HVM kernel-6.1

Architecture

64-bit (x86) ▼

Boot mode

uefi-preferred

AMI ID

ami-051f8a213df8bc089

Verified provider

Figure 5.8 – Selecting Amazon Linux

6. For **Instance type**, select **t2.micro**.

7. For **Key pair (login)**, either select the key pair name of a key pair we have already generated and have access to or, if we do not have a key pair, click on the **Create new key pair** link and complete the following steps:

A. Enter a key pair name.

B. Set **Key pair type** to **RSA**.

C. Set **Private key file format** to **.pem**.

D. Click on **Create key pair**.

Important note

We should save the key safely. If we are using a Unix or Mac system, then we need to change the file permission to read-only access with the `chmod 400` command.

8. In the **Network settings** section, click on **Edit** and do the following:
  - A. For **VPC**, select our VPC, which is named `awsseccb-vpc`.
  - B. For **Subnet**, select a public subnet in the `us-east-1a` AZ. If you have created a subnet as mentioned in the *Getting ready* section, these details are included in the name of the subnet.
  - C. Set **Auto-assign public IP** to **Enable**.

▼
Network settings
Info

VPC - required
Info

vpc-0c6aeafdd6dc7dc7d (awsseccb-vpc)
10.0.0.0/16

Subnet
Info

subnet-0207b1bee6e911ac9
awsseccb-subnet-public1-us-east-1a
VPC: vpc-0c6aeafdd6dc7dc7d
Owner: 207849759248
Availability Zone: us-east-1a
IP addresses available: 4091
CIDR: 10.0.0.0/20

Auto-assign public IP
Info

Enable

Additional charges apply when outside of free tier allowance

Figure 5.9 – The launch instance network settings

9. Under **Firewall (security groups)** in the **Networks** settings section, select **Create security group**. For **Security group name**, enter `cloudericks-web-server`. For **Description**, replace the default security group name within the **Description** field with `cloudericks-web-server`.

Firewall (security groups)
Info

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

☒ Create security group
☐ Select existing security group

Security group name - required

cloudericks-web-server

This security group will be added to all network interfaces. The name can't be edited after the security group is created. Max length is 255 characters. Valid characters: a-z, A-Z, 0-9, spaces, and \_-:/()#,@!+=&:~\*

Description - required
Info

cloudericks-web-server created 2024-04-17T19:07:00.778Z

Figure 5.10 Creating a security group



10. Under **Inbound Security Group Rules** within the **Network settings** section, add rules for **HTTP** and **HTTPS** with **Source type** as **Anywhere**. Add the **SSH** rule with the **Source type** as **My IP** to allow SSH only from our IP.

#### Important note

We allowed SSH traffic from our own IP address. In a production environment, SSH access is typically restricted to a bastion host or jump host, which is a dedicated server configured to provide a secure and controlled access point to the internal network from an external source. With a bastion host setup, we first log in to the bastion host and, from there, securely log in to our web server. We may also use one of the options shown such as EC2 Instance Connect, Session Manager, and EC2 serial console, as shown in *Figure 5.13*.

11. In the **Advanced details** section, copy and paste the following script code into the **User data** field:

```
#!/bin/bash
sudo su
yum update -y
yum install -y httpd
cd /var/www/html
echo "<html><h1>Cloudericks Web Server</h1></html>" > index.html
systemctl start httpd.service
systemctl enable httpd.service
```

12. Leave other values and selections as-is and click on **Launch instance**. Once the instance is launched successfully, go to the **Instances** page, select our new EC2 instance, and explore its parameters.
13. Copy either the **Public IPv4 DNS** or the **Public IPv4** address, open it from a browser tab, and use http instead of https. We should see our web server's index.html page.



## Cloudericks Web Server

Figure 5.11 – The web server's index.html page from a browser

In this section, we created a simple web server and accessed the web page from the internet using HTTP. It is listed as **Not Secure** since we are using HTTP and not HTTPS. If we try to run the URL with HTTPS, it will give a response similar to this:

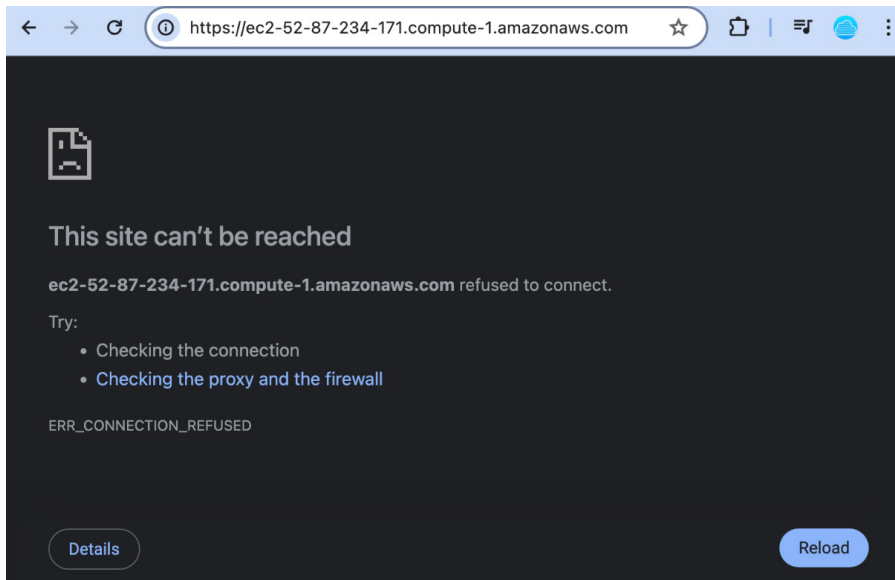


Figure 5.12 – No HTTPS website

In the next section, we will connect to the instance using SSH, and in *Chapter 6*, we will enable HTTPS on this machine.

### **Connecting to EC2 instance using SSH**

We can connect to the EC2 instance over SSH as follows:

1. Go to the **Instances** page in the **EC2** dashboard.
2. Select our instance and, from the **Actions** drop-down menu, click on **Connect**. This will give us ways to connect to an EC2 instance such as **EC2 Instance Connect**, **Session Manager**, **SSH client**, and **EC2 serial console**.

## Connect to instance [Info](#)

Connect to your instance i-05d82ab3459225afe (My Web Server) using any of these options


EC2 Instance Connect

Session Manager

SSH client

EC2 serial console

Instance ID


 i-05d82ab3459225afe (My Web Server)

Connection Type


☒ **Connect using EC2 Instance Connect**  
Connect using the EC2 Instance Connect browser-based client, with a public IPv4 address.

☐ **Connect using EC2 Instance Connect Endpoint**  
Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint.

Public IP address

 35.174.173.159

**Username**  
Enter the username defined in the AMI used to launch the instance. If you didn't define a custom username, use the default username, ec2-user.

 **Note:** In most cases, the default username, ec2-user, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.

Cancel

Connect

Figure 5.13 – The Connect to instance options


- Go to the **SSH client** tab. We should see the steps to connect to our instance using SSH, as shown in the following figure:



# Connect to instance [Info](#)

Connect to your instance i-05d82ab3459225afe (My Web Server) using any of these options


EC2 Instance Connect	Session Manager	<b>SSH client</b>	EC2 serial console
----------------------	-----------------	-------------------	--------------------

Instance ID

 **i-05d82ab3459225afe** (My Web Server)

1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is My EC2 Web Server Key Pair.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.  
 `chmod 400 "My EC2 Web Server Key Pair.pem"`
4. Connect to your instance using its Public DNS:  
 `ec2-35-174-173-159.compute-1.amazonaws.com`

Example:

 `ssh -i "My EC2 Web Server Key Pair.pem" ec2-user@ec2-35-174-173-159.compute-1.amazonaws.com`


 **Note:** In most cases, the guessed username is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.

Figure 5.14 – Connecting to an instance with the SSH client

4. We can follow the given steps and connect to our EC2 instance. If we are connecting to the instance for the first time, we will get a confirmation message to trust the site and continue connecting. Enter `yes` for the same:

[illegible]

Figure 5.15 – Connecting to the instance with SSH

We connected to our instance using SSH. It is worth noting that there are a few other options available, such as EC2 Instance Connect, Session Manager, and EC2 serial console, as we saw in *Figure 5.13*.

## How it works...

First, we set up a web server using EC2 user data. The EC2 user data feature allows us to configure a set of scripts or commands that will be run only once when the instance first starts. This capability is helpful for installing software, updating the system, downloading files, or configuring settings to match specific requirements. The user data script is injected into the instance before it is even booted, making it an efficient method to bootstrap the instance with all the necessary configurations and software. This helps to ensure that the instance is fully prepared for its role, whether it be a web server, database, or any other service, from the moment it becomes operational. This feature significantly streamlines the deployment process, eliminating the need for manual setup and allowing for more agile and scalable cloud architecture designs.

We also connected to our instance using SSH. SSH is a cryptographic network protocol employed for secure communication between a client and a server over an unsecured network. It's widely used for a variety of network services, with the most common being remote command-line login and execution. SSH provides a secure channel over an unsecured network, encrypting the data exchanged to prevent unauthorized access, eavesdropping, and hijacking. Aside from its primary function of secure remote administration, SSH also supports tunneling, forwarding **Transmission Control Protocol (TCP)** ports, and transferring files using associated protocols such as SFTP or SCP. Its versatility and security features make SSH an essential tool for managing servers, configuring networks, and securely transferring data across the internet.

## There's more...

We connected to our instance using SSH. AWS also provides a few alternatives to connect to our instance, as we saw in *Figure 5.14*. Let us quickly explore them.

EC2 Instance Connect provides a simple and secure way to connect to your instances using SSH directly from the AWS Management Console. Unlike traditional SSH, which requires you to manage SSH keys, Instance Connect handles key management for you, generating a one-time-use SSH key for each connection session. This method enhances security by avoiding the need to share and manage SSH keys manually and provides an easy way to control access through AWS IAM policies.

**AWS Systems Manager Session Manager** is a feature of AWS Systems Manager that lets you manage your EC2 instances through an interactive shell or automation scripts without needing to open inbound ports, set up a bastion host, or manage SSH keys. It provides secure, auditable instance management without the complexity of traditional access methods. Session Manager sessions are encrypted and can be logged and audited, making it an ideal choice for enterprises concerned with security and compliance. It also integrates with AWS IAM for access control.

The EC2 serial console allows you to troubleshoot boot and network connectivity issues by providing secure, serial access to your EC2 instances. This is particularly useful when you cannot connect to your instance using SSH or RDP. Access to the serial console does not require network connectivity, making it an invaluable tool for resolving issues that prevent an instance from starting up correctly. Access to the EC2 serial console is controlled through IAM policies, ensuring that only authorized users can use this feature.

Each of these connection methods serves different use cases, from simplifying SSH key management with EC2 Instance Connect, offering secure and auditable access with Session Manager, to providing a last-resort troubleshooting tool with the EC2 Serial Console. Depending on our security requirements, operational practices, and troubleshooting needs, we can choose the method that best fits our scenario.

## See also

- Read more about the different ways to connect to an EC2 instance at <https://www.cloudericks.com/blog/different-ways-to-connect-to-ec2-instances-in-aws>.
- We can further increase EC2 security by encrypting the EBS instance following the steps available at <https://www.cloudericks.com/blog/steps-to-encrypt-ebs-data-with-aws-kms>.

## Creating and configuring security groups

In this recipe, we will learn how to create a security group from the VPC dashboard. Similar steps can be followed to create a security group from the EC2 dashboard. We can also create a security group while launching an EC2 instance.

## Getting ready

To complete the steps within this recipe, we need the following configurations:

- A working AWS account is essential. I will be using the `awssecCb-sandbox-1` account that we created in *Chapter 1*. However, I will not be using any features of the AWS Organizations or the IAM Identity Center.
- We need to create a VPC and a public subnet under that. We can create a VPC and subnet by following the *Creating a bare VPC and setting up public and private subnets* recipe from this chapter.

## How to do it...

We can create a security group from the VPC dashboard as follows:

1. Go to the VPC dashboard.
2. On the left sidebar under **Security**, click on **Security groups**.
3. On the **Security Groups** page, click on **Create security group**.
4. Provide values for **Security group name** and **Description**; select our VPC.

## Create security group [Info](#)

A security group acts as a virtual firewall for your instance to control inbound and outbound traffic. To create a new security group, complete the fields below.

### Basic details

Security group name [Info](#)

AWSSecurityGroup

Name cannot be edited after creation.

Description [Info](#)

AWSSecurityGroup Description

VPC [Info](#)

vpc-0dedc90bd5ed1fd22 (MainVPC)

Figure 5.16 – Creating a security group

5. Under **Inbound rules** on the same page, we need to add the rules that follow:
- I. Click on **Add rule**. Set **Type** to **HTTP**, **Source type** to **Anywhere-IPv4**, and **Source** to 0.0.0.0/0.
  - II. Click **Add rule**. Set **Type** to **HTTPS**, **Source type** to **Anywhere-IPv4**, and **Source** to 0.0.0.0/0.
  - III. Click **Add rule**. Set **Type** to **SSH** and **Source type** to **My IP**. Our IP address should be populated under **Source**.

### Inbound rules [Info](#)

Type <a href="#">Info</a>	Protocol <a href="#">Info</a>	Port range <a href="#">Info</a>	Source <a href="#">Info</a>
HTTP	TCP	80	Anywhere... <div><div>0.0.0.0/0</div><div>0.0.0.0/0 X</div></div>
HTTPS	TCP	443	Anywhere... <div><div>0.0.0.0/0</div><div>0.0.0.0/0 X</div></div>
SSH	TCP	22	My IP <div><div>106.206.31.153/32</div><div>X</div></div>

Figure 5.17 – Setting the inbound rules for the security group

**Tip**  
If IPv6 traffic is required, we can also add a CIDR range of `:::/0` under **Source** to the rules.

6. Under **Outbound rules**, we can see a default rule already created as follows:

Outbound rules

Outbound rule 1

Delete

Type

Protocol

Port range

All traffic

All

All

Destination type

Destination

Description - optional

Custom

Q

0.0.0.0/0

Add rule

Figure 5.18 – The default outbound rule

**Important Note**  
Default outbound rules for a security group allow all outbound traffic. For added security, we can provide outbound access to only the required protocols, such as HTTP and HTTPS.

If we need to change the outbound rules so that we only allow HTTP and HTTPS traffic, edit the default rule on the same page as follows:

- I. Set **Type** to **HTTP**, **Destination type** to **Anywhere-IPv4**, and **Destination** to `0.0.0.0/0`.
- II. Click **Add rule**. Set **Type** to **HTTPS**, **Destination type** to **Anywhere-IPv4**, and **Destination** to `0.0.0.0/0`.

Outbound rules

Type

Protocol

Port range

Destination

HTTP

TCP

80

Anywhe...

HTTPS

TCP

443

Anywhe...

Figure 5.19 – Outbound rules for the security group



**Tip**

If IPv6 traffic is required, we can also add a CIDR range of `:::/0` under **Destination** to the rules.

7. Scroll down and click **Create security group**.

We should get a success message stating that the security group was created.

**Important note**

The exact rules will be different for each recipe. The preceding rules may be used for instances of a public subnet that hosts a web server. We also provided SSH access to our local IP; however, in most projects, we would give SSH access to a dedicated machine, referred to as a jump host or a bastion host.

## How it works...

In this recipe, we created and configured a security group with inbound and outbound rules that are applicable to an EC2 instance in a public subnet running a web server. We will use these steps to create security groups in other recipes. The exact rules may differ based on the use case. Instead of providing the CIDR range, we can also specify another security group in a rule to say that only instances with that security group should be allowed.

In the *Working with NACLs* recipe later in this chapter, we will explicitly allow the 1024 – 65535 **ephemeral port** range for outbound requests. This isn't needed for security groups since security groups are stateful. If an outbound port is opened, the response for a request going through that port is also allowed, irrespective of the inbound rules. Similarly, if an inbound port is opened, the response for a request coming through that port is also allowed, irrespective of the outbound rules.

## There's more...

We created security groups from the VPC dashboard. We can also create them from the EC2 dashboard. Refer to the links within the *See also* section of this recipe for more details.

Let's quickly go through some important concepts related to security groups:

- Security groups do not span across VPCs.
- We can create security groups from the EC2 launch wizard, EC2 dashboard, or VPC dashboard.
- Security groups are stateful, unlike NACLs.
- It is good practice to have multiple security groups based on usage. For example, we can create separate security groups: one for SSH and one for application-specific ports. We can configure the rules for a security group to allow instances from another security group instead of providing a CIDR. We can also specify our own security group to allow only instances within the same security group to talk to each other.

## See also

- Read more about security groups in AWS at <https://www.cloudericks.com/blog/understanding-security-groups-in-aws>.
- A rules reference for security groups can be found at <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/security-group-rules-reference.html>.

## Working with NACLs

In this recipe, we will create a new NACL with no SSH support and associate one of our subnets with that NACL. By doing this, we will see that we cannot do SSH into EC2 instances within that subnet. After, we will add SSH support to the NACL and try to SSH again.

## Getting ready

To complete the steps within this recipe, we need the following:

- We need to create a VPC and a public subnet under that. We can create a VPC and subnet by following the *Creating a bare VPC and setting up public and private subnets* recipe from this chapter.
- We need to create a security group for our instance; this security group should allow inbound traffic for SSH. We can do this by referring to the *Creating and configuring security groups* recipe in this chapter.
- We need an EC2 instance launched into a public subnet; we can do this by referring to the *Launching an EC2 instance with a web server using user data* recipe in this chapter.

## How to do it...

We can create an NACL with no SSH permission as follows:

1. Go to the **VPC** service in the console.
2. Click on **Network ACLs** on the left sidebar.
3. Click on **Create network ACL** at the top of the page.
4. In **Create network ACL** under **Network ACL settings**, provide a name and select the VPC we created in the previous section from the dropdown for the **VPC** field.
5. Click on **Create network ACL** to create the network ACL.

### Network ACL settings

**Name - optional**  
Creates a tag with a key of 'Name' and a value that you specify.

MyNACLs

**VPC**  
VPC to use for this network ACL.

vpc-0e114addc21edc8f9 (Myvpc99) ▼

### Tags

A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.

Key

Value - optional

Q Name X

Q MyNACLs X

Remove tag

Add tag

You can add 49 more tags

Cancel

Create network ACL

Figure 5.20 – Creating a network ACL

If we go to the NACL list, we will see that our new NACL doesn't have any subnets associated with it.

Network ACLs (5) Info

Find resources by attribute or tag

Actions

Create network ACL

< 1 >

<input type="checkbox"/>	Name	Network ACL ID	Associated with	Default	VPC ID
<input type="checkbox"/>	MyNACL	acl-0eaaee394335443ef	-	No	vpc-09391481268ff8119 / Public

Figure 5.21 – A new NACL is created without any associated subnets

6. Select our new NACL, scroll down, and verify the inbound and outbound rules of the new NACL from its **Inbound Rules** and **Outbound Rules** tabs, respectively.

The inbound rules should be as follows:

acl-0eaaee394335443ef / MyNACL

Details | **Inbound rules** | Outbound rules | Subnet associations | Tags

**Inbound rules (1)** Edit inbound rules

Q Filter inbound rules

Rule number	Type	Protocol	Port range	Source	Allow/Deny
*	All traffic	All	All	0.0.0.0/0	⊗ Deny

Figure 5.22 – The inbound rules for the new NACL created

The outbound rules should be as follows:

acl-0eaaee394335443ef / MyNACL

Details | **Inbound rules** | **Outbound rules** | Subnet associations | Tags

**Outbound rules (1)** Edit outbound rules

Q Filter outbound rules

Rule number	Type	Protocol	Port range	Destination	Allow/Deny
*	All traffic	All	All	0.0.0.0/0	⊗ Deny

Figure 5.23 – The outbound rules for the new NACL created

7. Click on the **Subnet associations** tab.
8. Click on **Edit subnet associations**.
9. Select the subnet we created (AWSPublicSubnet) and click **Save changes**.

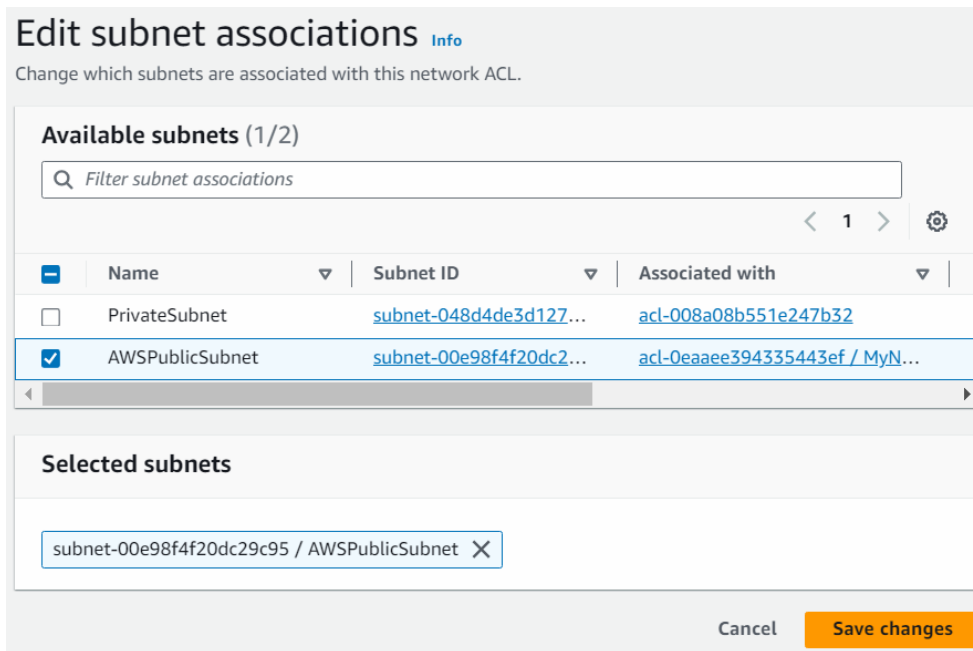


Figure 5.24 – Associating subnets

Select our new network ACL and check its subnet associations. Our public subnet should now be associated with it.

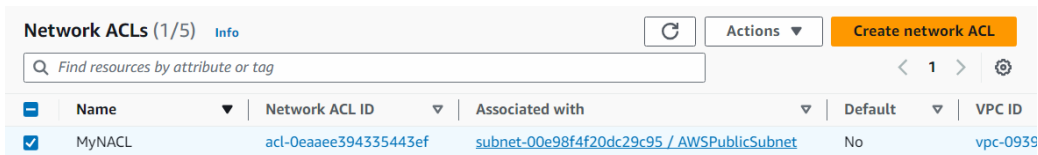


Figure 5.25 – The NACL with the associated subnets

10. Try to SSH into our public EC2 instance. Run the command that follows in the CLI:

```
ssh -i "<key path>" <EC2 user name@the public ip>
```

The `<key path>` path is the path to the key pair that you had downloaded as part of the *Launching an EC2 instance with a web server using user data* recipe mentioned in the *Getting ready* section of this recipe. For `<EC2 user name@the public ip>`, go to **Instances**, select the EC2 instance we created, and click on **Connect**. Click on the **SSH client** tab. Copy the second part of the **Example** command like `ec2-user@54.198.244.252`. It will look similar to the following:

```
ssh -i "C:\Users\DELL\Downloads\awsdemo.pem"
ec2-user@35.174.184.142
```

**Tip**

If we are using AWS CloudShell, we can use the **Example** command from the **SSH client** tab without any modification. It will look similar to `ssh -i "awsdemo.pem" ec2-user@35.174.184.142`. Make sure to upload the key pair file before running the command.

The exact command or steps may differ between operating systems. On Windows, macOS, and most Linux systems, we can use the SSH command.

The operation should time out, as shown in the following screenshot:

```
C:\Users\DELL>ssh -i "C:\Users\DELL\Downloads\awsdemo.pem" ec2-user@35.174.184.142
ssh: connect to host 35.174.184.142 port 22: Connection timed out
```

Figure 5.26 – A connection time-out response without SSL

11. Now, we can add SSH support to our NACL as follows:

- I. Go back to the VPC dashboard, click on **Network ACLs** from the left sidebar, and select our NACL.
- II. Click on **Inbound rules**.
- III. Click on **Edit inbound rules**.
- IV. Click on **Add new rule**.
- V. Enter 100 under **Rule number**, select **SSH (22)** under **Type**, leave the source as 0.0.0.0/0, set **Allow/Deny** to **Allow**, and click on **Save changes**.

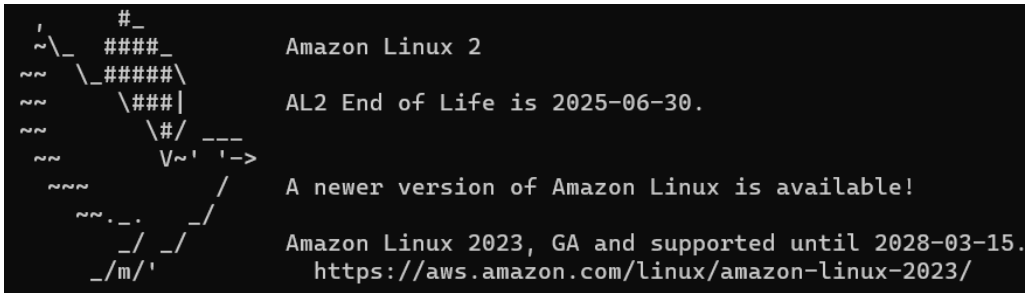
**Important note**

If we try to SSH into the EC2 instance now, the SSH will fail, as we have not enabled the ephemeral ports for outbound traffic.

12. Click on **Outbound rules**.
13. Click on **Edit outbound rules**.
14. Click on **Add new rule**.
15. Enter 100 under **Rule number**, select **Custom TCP** under **Type**, set **Port range** to 1024 - 65535, set **Allow/Deny** to **Allow**, set **Destination** to 0.0.0.0/0, and click on **Save changes**.
16. Try to SSH into our public EC2 instance. The exact command or steps may differ between operating systems. On Windows, macOS, and most Linux systems, we can use the SSH command that follows:

```
ssh -i "C:\Users\DELL\Downloads\awsdemo.pem"
ec2-user@35.174.184.142
```

Now, we should be able to SSH successfully.



```
#_
Amazon Linux 2
AL2 End of Life is 2025-06-30.
A newer version of Amazon Linux is available!
Amazon Linux 2023, GA and supported until 2028-03-15.
https://aws.amazon.com/linux/amazon-linux-2023/
```

Figure 5.27 – A successful SSH response

In this recipe, we only added one inbound rule and one outbound rule. We can add more rules as required.

## How it works...

NACLs allow us to define inbound and outbound rules for the subnets of our VPC. We can explicitly allow or deny traffic through a port, or a range of ports. The default NACL that was created by AWS allows all inbound and outbound traffic. However, by default, a custom NACL denies all inbound and outbound traffic. First, we created a new NACL. Then, we associated our public subnet with that NACL and verified that we could not SSH from our local machine. A new NACL denies inbound and outbound traffic by default. To allow SSH, we added an inbound rule for SSH in our NACL and an outbound rule to allow the 1024 – 65535 ephemeral port range.

### Important Note

An ephemeral port is a short-lived port for IP communications with transport protocols such as TCP, **User Datagram Protocol (UDP)**, **Stream Control Transmission Protocol (SCTP)**, and so on. It is usually used for the return traffic from the instance or service we are connecting to. For example, the server accepts SSH traffic on port 22 and then communicates to the client through one of the ephemeral ports. In this recipe, we added outbound rules that allow the ephemeral port range as suggested by AWS for public-facing instances to cover various client types.

## There's more...

Let's quickly go through some important concepts related to network ACLs:

- When we create a VPC, a default NACL is created by AWS. The value of the **Default** column will be **Yes** for the default NACL within the NACL list in our VPC.
- Default NACL allows all inbound and outbound traffic. However, when we create a new custom NACL, all inbound and outbound traffic is denied by default.

- Every subnet needs to be associated with one NACL at a time. By default, a subnet is associated with the default NACL.
- One subnet can only be associated with one NACL at a time. When we associate it with a new NACL, the current association will be removed.
- A single NACL can be associated with multiple subnets.
- NACLs contain a numbered set of rules. These rules are evaluated in the order of the rule numbers. If we have an **Allow** rule before a **Deny** rule for the same port, access will be allowed for that port. Similarly, if we have a **Deny** rule before an **Allow** rule for the same port, access will be denied for that port. AWS recommends using rule numbers in multiples of 100 initially as that will let us add new rules in between if needed.
- We can block specific IP addresses with NACL, but this is not possible with security groups.
- NACLs are evaluated before security groups.
- Security groups are considered stateful, while NACLs are considered stateless. With a security group, if we send a request from the instance, the response is allowed, irrespective of the inbound rules. Similarly, if we allow an inbound request, the corresponding outbound response can go, regardless of the outbound rules. With NACL, we need to allow both inbound and outbound traffic explicitly for any port.

## See also

Read more about NACLs at <https://www.cloudericks.com/blog/understanding-network-acls-in-aws-vpc>.

## Using a VPC gateway endpoint to connect to S3

In this recipe, we will create a **VPC gateway endpoint** for S3 and connect to S3 from our private subnet without any internet access.

## Getting ready

To complete the steps within this recipe, we need to have the following ready:

- We need a VPC with associated subnets. We can create one by referring to the *Creating a bare VPC and setting up public and private subnets* recipe from this chapter.
- We need to configure a gateway and a route table for internet access. We may refer to the *Creating a bare VPC and setting up public and private subnets* recipe from this chapter.
- Subnets should be associated with the default NACL. Otherwise, we should define proper inbound and outbound rules so that we can log in to the private EC2 instance through the public EC2 instance. We can refer to the *Working with NACLs* recipe from this chapter.



- We need an S3 bucket in any region. I will be using `us-east-1`.
- We should have no internet access for the private subnet. Verify this by running `aws s3 ls --region us-east-1` from our private subnet. Our requests should fail with a timeout. If a NAT gateway or a NAT instance has been configured, remove its route from the main route table.
- Associate an IAM role with S3 access to a private EC2 instance.

**Tip**

If you have not configured an IAM role correctly, you might get an error that says **Unable to locate credentials**. You can configure credentials by running `aws configure`. Fix the issue and test again before proceeding.

**How to do it...**

We can create a VPC endpoint gateway for S3 as follows:

1. Go to the **VPC** service in the console.
2. Click on **Endpoints** from the left sidebar.
3. Click on **Create endpoint**.
4. In the **Create endpoint** pane, under **Endpoint settings**, enter `VPCEndpoint` as the value for **Name tag**.
5. Under **Service category**, select **AWS services**.
6. Under **Service Name**, select `com.amazonaws.us-east-1.s3`

	Service Name ▾	Owner ▾	Type
<input checked="" type="radio"/>	com.amazonaws.us-east-1.s3	amazon	Gateway
<input type="radio"/>	com.amazonaws.us-east-1.s3	amazon	Interface

Figure 5.28 – Selecting the service name for endpoint creation

7. For **VPC**, select the VPC we created in the *Getting ready* section.
8. For **Route tables**, select the route table we created in the *Getting ready* section.
9. Leave the **Policy** field set to **Full access**.
10. Click on **Create endpoint**. We should get a success message.
11. Connect to our EC2 instance and try running the following S3 command from the private subnet:

```
aws s3 ls --region us-east-1
```

```
[ec2-user@ip-10-0-6-72 ~]$ aws s3 ls --region us-east-1
2023-11-23 09:48:12 0987654321qoiyhoyty78t
2023-11-06 15:53:54 amplify-petstoresample-devf-155348-deployment
2023-11-28 05:43:13 awsrecipebucket
2023-11-24 11:54:44 awsseccookbook2
2023-11-25 02:47:36 awsseccookbookbucket
2023-11-21 03:54:33 awsseccookbookmumbai
2023-11-29 06:29:46 cloudfrontbucketdemo1
2023-11-29 08:57:00 demo1cloudfronts3bucket
2023-11-25 10:53:42 demobucket098
2023-11-25 11:51:12 demopracticebucket123
2023-11-28 11:28:25 demousersbucket
2023-12-05 14:03:32 endpoint123bucket
2023-11-24 05:29:01 sivabucketonly123
2023-11-22 06:13:20 sourcebucketsiva
```

Figure 5.29 – A successful response for S3 list operation

This should list the S3 items successfully.

## How it works...

VPC endpoints allow us to connect to supported AWS services from our VPC privately. With VPC endpoints, instances in the VPC do not need a public IP address to communicate with supported AWS services. The traffic between our VPC and the supported AWS services does not leave AWS. VPC endpoints can be considered highly available virtual devices.

In this recipe, we configured a VPC endpoint of the gateway endpoint type to access S3 from our subnet. We removed all public routes from our subnet, and we could still connect to S3. VPC gateway endpoints are also supported by **DynamoDB** and work similarly to a VPC gateway. For most other services, VPC endpoints are supported through interface endpoints.

## There's more...

Let's quickly go through some important concepts related to VPC endpoints. There are two types of VPC endpoints:

- **Interface endpoints:** This is an **Elastic Network Interface (ENI)** with a private address that allows traffic to a supported service. There are around 20 supported services. Examples of such supported services include Amazon API Gateway, Amazon CloudWatch, AWS Config, AWS KMS, and so on.
- **Gateway endpoints:** Like NAT gateways, they do not have private IP addresses. This is only supported for limited services such as S3 and DynamoDB.

## See also

Read more about the VPC gateway endpoint at <https://www.cloudericks.com/blog/understanding-aws-vpc-gateway-endpoint>.

## Configuring and using VPC flow logs

In this recipe, we will enable flow logs at the VPC level.

### Getting ready

We need the following resources for completing the steps within this recipe:

- A **CloudWatch log group** will be needed. The detailed steps are provided later in this section.
- We need to set up a VPC. If one hasn't been created previously, please refer to the *Creating a bare VPC and setting up public and private subnets* recipe.
- An IAM role with permissions to publish to the CloudWatch log group with full access will also be needed.

We can perform the following steps to create a CloudWatch log group:

1. Go to the CloudWatch service in the AWS console.
2. Click on **Logs** from the left sidebar.
3. Click on **Log groups** and click on **Create log group**.
4. Give the log group a name that describes its purpose, keep the other values as their defaults, and click on **Create**.

### How to do it...

We can configure VPC flow logs from the console as follows:

1. Go to the **VPC** service in the console.
2. Click on **Your VPCs**.
3. Select our VPC.
4. Click on the **Flow Logs** tab.
5. Click on the **Actions** dropdown and select **Create flow log**.
6. Under **Flow log settings**, provide a name, and for the **Filter**, select **All**.
7. For **Maximum aggregation interval**, select the time based on your preferences.
8. Set **Destination** to **Send to CloudWatch Logs**.

9. Select **Destination log group** as the log group we created in the *Getting ready* section of this recipe.
10. Select the option we created in the *Getting ready* section of this recipe from the drop-down list for the **IAM role**.
11. For **Log record format**, select **AWS default format**.
12. Click on **Create flow log**. We should see a success message. We should be able to see all further IP traffic logs within our flow logs. The following is an example of a log record from the log group for VPC logs:

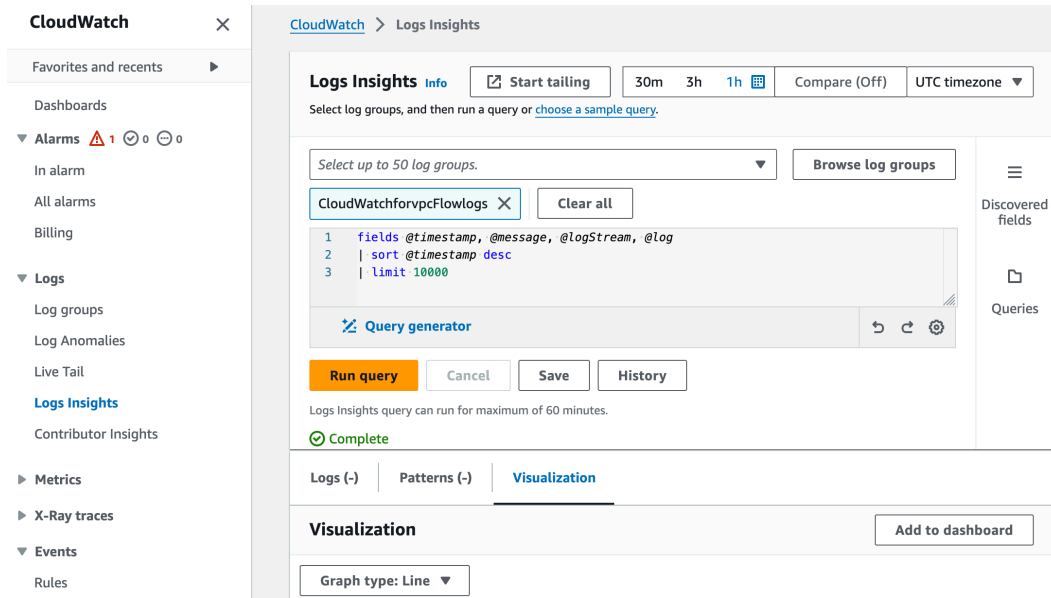


Figure 5.30 – A sample log record

## How it works...

VPC flow logs help us capture IP traffic to and from our VPCs. Data from VPC flow logs can be published to either CloudWatch logs or to an S3 bucket. We can choose to log only accepted traffic, rejected traffic, or both. VPC flow logs can be created at different levels, such as the VPC level, subnet level, and network interface level.

In the recipe, within the **Filter** dropdown, we selected **All** to log all IP traffic to and from our VPCs. We can choose **Accept** to log only accepted traffic, **Reject** to log only rejected traffic, and **All** to log both accepted and rejected traffic. We needed a CloudWatch log group and an IAM role with permission to log to that log group.

## There's more...

Let's quickly go through some more important concepts related to flow logs:

- Currently, we cannot change a flow log configuration, such as changing the associated IAM role, once it's been created.
- Some of the IP traffic, including the ones listed here, are not monitored by flow logs:
  - Traffic to the reserved IP addresses of the default VPC router
  - **Dynamic Host Configuration Protocol (DHCP)** traffic
  - Traffic set to 169.254.169.254 for querying instance metadata
  - Traffic while contacting Amazon DNS servers via instances; however, traffic to our own DNS server is logged
  - Windows license activation traffic

## See also

- Read more about VPC flow logs at <https://www.cloudericks.com/blog/understanding-aws-vpc-flow-logs>.
- Find examples of log records at <https://docs.aws.amazon.com/vpc/latest/userguide/flow-logs-records-examples.html>.

## Setting up and configuring NAT gateways

In this recipe, we will learn how to create and configure NAT gateways, which is the latest and preferred option for NAT in AWS.

### Getting ready

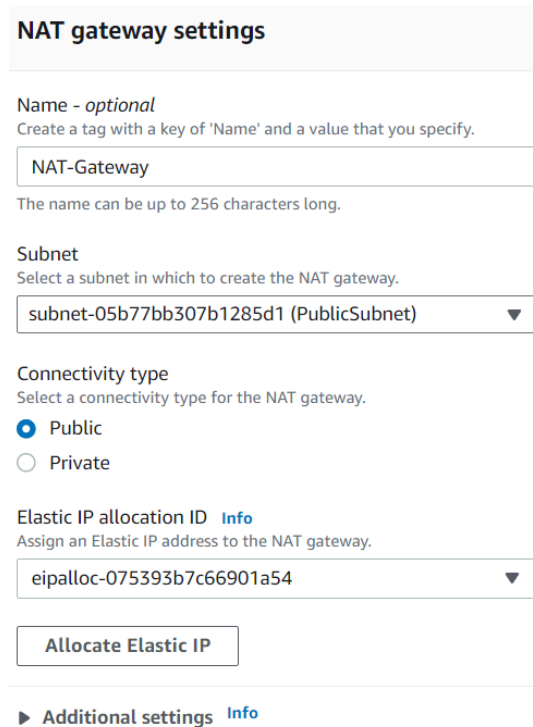
To complete the steps within this recipe, we need the following:

- A working AWS account is essential. I will be using the `awsseccb-sandbox-1` account that we created in *Chapter 1*.
- Create a VPC, subnets, an IGW, and a route table by following the *Creating a bare VPC and setting up public and private subnets* recipe from this chapter.
- Launch instances in the public and private subnets with the appropriate security group configurations by following the *Creating and configuring a security group* recipe and the *Getting ready* section of the *Working with NACLs* recipe in this chapter.

## How to do it...

We can create a NAT gateway as follows:

1. Go to **VPC dashboard**.
  2. Click on **NAT gateways** on the left pane.
  3. Click on **Create NAT gateway**.
  4. Under the **NAT gateway settings**, type NAT-Gateway under **Name**.
  5. For **Subnet**, in the drop-down list, select our public subnet.
  6. For **Connectivity type**, select **Public**.
  7. Click on **Allocate Elastic IP** to populate the allocation ID for an Elastic IP and select it for **Elastic IP allocation ID** or, if we already have an Elastic IP created, we can select one of them.
- The filled page should look as follows:



The screenshot shows the 'NAT gateway settings' form. It includes a text input for 'Name' with the value 'NAT-Gateway', a dropdown for 'Subnet' with the selected value 'subnet-05b77bb307b1285d1 (PublicSubnet)', and radio buttons for 'Connectivity type' with 'Public' selected. Below these is a dropdown for 'Elastic IP allocation ID' with the value 'eipalloc-075393b7c66901a54'. At the bottom of the settings section is a button labeled 'Allocate Elastic IP'. Below the settings section is a link for 'Additional settings'.

**NAT gateway settings**

Name - *optional*  
Create a tag with a key of 'Name' and a value that you specify.

NAT-Gateway

The name can be up to 256 characters long.

Subnet  
Select a subnet in which to create the NAT gateway.

subnet-05b77bb307b1285d1 (PublicSubnet) ▼

Connectivity type  
Select a connectivity type for the NAT gateway.

☒ Public  
☐ Private

Elastic IP allocation ID [Info](#)  
Assign an Elastic IP address to the NAT gateway.

eipalloc-075393b7c66901a54 ▼

[Allocate Elastic IP](#)

► [Additional settings](#) [Info](#)

Figure 5.31 – NAT gateway settings

8. Click on **Create a NAT gateway**. We will get a message saying that the NAT gateway was created successfully.

9. Click on **Route tables**, select our public route table, and go to the **Routes** tab. Click on **Edit routes** and then on **Add route**. Select `0.0.0.0/8` as the **Destination**. From the dropdown for **Target**, select **NAT Gateway** and then select the NAT gateway we created in *Step 8*. Click on **Save changes**.
10. Go back to EC2 instances, select our instance, click on **Connect**, then on the **Connect to instance** page, click on **Connect** again.
11. Try running any command from the terminal that requires internet access, as follows:

```
ping google.com
```

We should get back a successful response if there is a route to the internet; otherwise, it will time out:

```
[ec2-user@ip-10-0-21-109 ~]$ ping google.com
PING google.com (142.250.31.139) 56(84) bytes of data.
64 bytes from bj-in-f139.1e100.net (142.250.31.139): icmp_seq=1 ttl=58 time=1.74 ms
64 bytes from bj-in-f139.1e100.net (142.250.31.139): icmp_seq=2 ttl=58 time=1.79 ms
64 bytes from bj-in-f139.1e100.net (142.250.31.139): icmp_seq=3 ttl=58 time=1.74 ms
64 bytes from bj-in-f139.1e100.net (142.250.31.139): icmp_seq=4 ttl=58 time=1.78 ms
64 bytes from bj-in-f139.1e100.net (142.250.31.139): icmp_seq=5 ttl=58 time=1.76 ms
64 bytes from bj-in-f139.1e100.net (142.250.31.139): icmp_seq=6 ttl=58 time=1.78 ms
64 bytes from bj-in-f139.1e100.net (142.250.31.139): icmp_seq=7 ttl=58 time=1.78 ms
64 bytes from bj-in-f139.1e100.net (142.250.31.139): icmp_seq=8 ttl=58 time=1.79 ms
64 bytes from bj-in-f139.1e100.net (142.250.31.139): icmp_seq=9 ttl=58 time=1.75 ms
^C
--- google.com ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8015ms
rtt min/avg/max/mdev = 1.745/1.774/1.798/0.059 ms
[ec2-user@ip-10-0-21-109 ~]$
```

Figure 5.32 – A successful response to the ping command

We can also run a yum update command as follows:

```
sudo yum update
```

Follow the remaining prompts. The update will happen successfully if there is a route to the internet; otherwise, it will time out.

## How it works...

Internet access may be required for instances in our private subnets for activities such as patching, downloading software, and so on. NAT allows a private subnet in our VPC to talk to the internet. NAT is a process of remapping the IP address of a packet by modifying its IP header while in transit. AWS provides us with two ways to achieve NAT with VPCs: NAT gateways and NAT instances. We created and configured a NAT gateway in this recipe. Unlike NAT instances, NAT gateways are not associated with any security groups and hence, we did not create or configure any security groups.

After creating the NAT gateway, we need to add a route for it within the route table that our private subnets are associated with. We added the route to our main route table since our private subnets are associated with the main route table. A subnet that is not associated explicitly with any route table will be implicitly associated with the main route table. If our architecture has a different route table for private subnets, we will need to add the route for the NAT gateway within that route table.

## There's more...

Let's quickly go through some more important concepts related to NAT gateways:

- NAT gateways are maintained by AWS and AWS takes care of patching, availability, and scaling.
- NAT gateways are not associated with any security groups. NAT gateways are redundant within an AZ, but cannot span an AZ. Therefore, for better availability, we may need to create a NAT gateway per region.
- NAT is currently not supported for IPv6 traffic. We need to use an egress-only IGW instead of NAT for IPv6 traffic. We can create an egress-only IGW from the VPC dashboard.

## See also

- Read more about NAT gateways at <https://www.cloudericks.com/blog/understanding-nat-gateways-in-aws>.
- Read about how we can securely connect our VPC to supported AWS services, other VPCs, and on-premises applications, without exposing our traffic to the public internet using PrivateLink, at <https://www.cloudericks.com/blog/understanding-aws-privatelink>.





# 6

## Web Security Using Certificates, CDNs, and Firewalls

In today's interconnected digital landscape, web security is paramount. To maintain the integrity and confidentiality of data, it is crucial to use a combination of tools and technologies. This chapter focuses on three key components – **certificates**, **Content Delivery Networks (CDNs)**, and **firewalls**. We will also see how we can use certificates alongside load balancers. Certificates, particularly X.509, play a vital role in securing communications between clients and servers by enabling **TLS** (short for **Transport Layer Security**). This encryption helps to safeguard data in transit, making it essential for preventing data breaches and maintaining privacy.

**Load balancers** enhance the reliability and performance of web services by efficiently distributing incoming network traffic across multiple servers. By doing so, they ensure that no single server bears too much load, thus preventing downtime and optimizing resource use. Load balancers when used with certificates offer the flexibility of managing security measures such as TLS termination, allowing for centralized handling of certificates, which simplifies security management and boosts performance.

CDNs improve website load times and reduce bandwidth costs by caching content at multiple geographic locations, closer to users. They also add a layer of security by protecting against **Distributed Denial of Service (DDoS)** attacks and enhancing the availability and performance of web applications.

Lastly, firewalls are critical in defining and enforcing the security perimeter of your network. They monitor and control the flow of traffic to and from your network. When integrated within environments such as AWS, firewalls help create a robust defense against potential threats, contributing significantly to the overall security strategy. This chapter will explore practical implementations of these components within AWS, providing you with the knowledge to build a secure, scalable web infrastructure.

In this chapter, we will cover the following recipes:

- Enabling HTTPS for a web server on an EC2 instance
- Creating an SSL/TLS certificate with ACM
- Creating ELB target groups
- Using an application load balancer with TLS termination at the ELB
- Using a network load balancer with TLS termination at EC2
- Securing S3 using CloudFront and TLS
- Using a WAF

## Technical requirements

Before diving into the recipes of this chapter, we need to ensure we have the following requirements and knowledge in place:

- We need an active AWS account to complete the recipes within this chapter. We can use an account that is part of an AWS organization or a standalone account. I will be using the `awssecb-sandbox-1` account that we created in the *Multi-account management with AWS Organizations* recipe in *Chapter 1*. However, I won't be utilizing any AWS Organizations features, meaning you can follow these steps with a standalone account too.
- For administrative actions, we need a user who has `AdministratorAccess` permission to the AWS account we will work with. This can be an IAM Identity Center user or an IAM user. I will be using `awssecbadmin1`, the IAM Identity Center user we created in the *User management and SSO with IAM Identity Center* recipe in *Chapter 1*. However, I won't be utilizing any IAM Identity Center features, meaning you can follow these steps as an IAM user, too, if the user has `AdministratorAccess` permission within the account. You can create an IAM user by following the *Setting up IAM, account aliases, and billing alerts* recipe in *Chapter 1*.

The code files for this book are available at <https://github.com/PacktPublishing/AWS-Security-Cookbook-Second-Edition>. The code files for this chapter are available at <https://github.com/PacktPublishing/AWS-Security-Cookbook-Second-Edition/tree/main/Chapter06>.

## Enabling HTTPS for a web server on an EC2 instance

In the *Launching an EC2 instance with a web server using user data* recipe in *Chapter 5*, we launched an EC2 instance with a web server without enabling HTTPS. In this recipe, we will demonstrate how to enable HTTPS on that web server using a self-signed certificate. This will help us to understand the fundamental concepts of enabling HTTPS on an EC2 instance. However, for practical applications, it is advisable to use methods outlined in other recipes within this chapter, which involve certificates signed by a **Certificate Authority (CA)**, such as **AWS Certificate Manager (ACM)**.

### Getting ready

We need the following to successfully complete this recipe:

- A working AWS account, `awsseccb-sandbox-1`, and a user, `awsseccbadmin1`, as described in the *Technical requirements* section.
- An EC2 instance called `Cloudericks Web Server`, launched in the *Launching an EC2 instance with a web server using user data* recipe from *Chapter 5*.

### How to do it...

We will enable HTTPS using a self-signed certificate, as follows:

1. Install the Apache `mod_ssl` module:

```
sudo yum install -y mod_ssl
```

2. Generating a private key is the first step in setting up HTTPS encryption for a web server. The private key is a crucial component of the SSL/TLS encryption process. Use the following command to generate a private key:

```
sudo openssl genrsa -out /etc/pki/tls/private/localhost.key 2048
```

This will return nothing but generate a 2,048-bit RSA private key, saved in `/etc/pki/tls/private/localhost.key` file.

3. Generate a **certificate signing request (CSR)** with the private key generated in the previous step, using the following command:

```
sudo openssl req -new -key /etc/pki/tls/private/localhost.key  
-out /etc/pki/tls/certs/localhost.csr
```

4. After running the command from the previous step in the command line, when prompted, enter the values as shown in the following figure:

```
[ec2-user@ip-172-31-83-105 ~]$ sudo openssl req -new -key /etc/pki/tls/private/localhost.key
-out /etc/pki/tls/certs/localhost.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:IN
State or Province Name (full name) []:KERALA
Locality Name (eg, city) [Default City]:KOTTAYAM
Organization Name (eg, company) [Default Company Ltd]:TRAINS0
Organizational Unit Name (eg, section) []:CLOUDERICKS
Common Name (eg, your name or your server's hostname) []:HEARTIN
Email Address []:heartin@cloudericks.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:CLOUDERICKS#7411174112
String too long, must be at most 20 bytes long
A challenge password []:CLDRCKS#74111
An optional company name []:TRAINS0
[ec2-user@ip-172-31-83-105 ~]$
[ec2-user@ip-172-31-83-105 ~]$
[ec2-user@ip-172-31-83-105 ~]$
```

Figure 6.1 – The certificate signing request

5. Use the CSR generated in the previous step to create a self-signed certificate:

```
sudo openssl x509 -req -days 365 -in /etc/pki/tls/certs/
localhost.csr -signkey /etc/pki/tls/private/localhost.key -out /
etc/pki/tls/certs/localhost.crt
```

```
[ec2-user@ip-172-31-83-105 ~]$ sudo openssl x509 -req -days 365 -in
/etc/pki/tls/certs/localhost.csr -signkey /etc/pki/tls/private/local
host.key -out /etc/pki/tls/certs/localhost.crt
Certificate request self-signature ok
subject=C = IN, ST = KERALA, L = KOTTAYAM, O = TRAINS0, OU = CLOUDER
ICKS, CN = HEARTIN, emailAddress = heartin@cloudericks.com
[ec2-user@ip-172-31-83-105 ~]$
```

Figure 6.2 – Create a self signed certificate

When we run this command, OpenSSL reads the CSR, signs it using the private key, and generates a self-signed certificate (`localhost.crt`). This certificate can then be used by your web server (e.g., Apache) to enable HTTPS encryption and establish secure communication with clients. The certificate will be in the `/etc/pki/tls/certs/localhost.crt` location.

- Restart the Apache server with the following command:

```
sudo systemctl restart httpd
```

- Access the URL of our web server using the HTTPS prefix, as demonstrated in the last step of the *Setting up a web server using EC2 user data* section in the *Launching an EC2 instance with a web server using user data* recipe in *Chapter 5*. We should now receive a response, but a **Not Secure** warning may also appear, as we used a self-signed certificate.

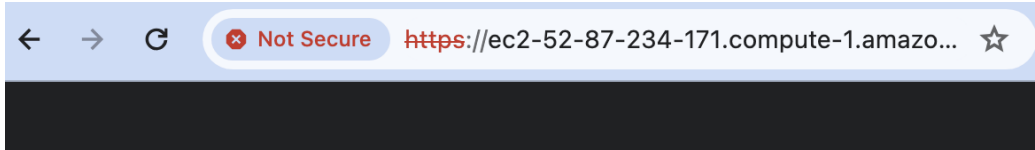


Figure 6.3 – A HTTPS URL with a self-signed certificate

In this section, we enabled HTTPS for our web server on our EC2 instance using a self-signed certificate. In the next recipe, we will explore how to enhance security by using a certificate signed by a CA.

## How it works...

First, we installed the Apache `mod_ssl` module. This module adds TLS support to our Apache server. The `mod_ssl` module is required to enable SSL/TLS support in Apache, and it needs to be installed and configured before generating the self-signed certificate. The current version of the module also supports SSL v3 and all versions of TLS. First, generate a private key using OpenSSL. Then, create a **certificate signing request (CSR)** and generate a self-signed certificate using the key. Restart Apache to apply the changes and allow HTTPS traffic through the firewall. Test HTTPS access in a web browser.

## There's more...

In this recipe, we used a self-signed certificate and encountered a **Not Secure** warning, as shown in *Figure 6.3*. To address this, we can generate a certificate through a CA. A CA is an entity that issues digital certificates to organizations or individuals after validating them. By obtaining a certificate from a CA, we can ensure that the SSL/TLS certificate is trusted by most web browsers and users' devices, eliminating warnings about insecure connections. CAs play a critical role in how the internet operates and how secure, encrypted communications are established.

There are several types of CAs, ranging from large, widely recognized organizations that issue certificates to the public to smaller, private CAs that might be internal to an organization for its own use. Public CAs such as Verisign, Comodo, and DigiCert provide extensive validation services, including **Extended Validation (EV)** and **Organization Validated (OV)** certificates that offer higher levels of security and trust. For websites handling sensitive transactions, these certificates can enhance credibility and user trust.

Cloud service providers such as AWS, Microsoft Azure, and Google Cloud offer integrated SSL/TLS certificate management solutions that simplify the provisioning, management, and deployment of certificates within their ecosystems. ACM is ideal for AWS-native services, automating renewal and management, but it restricts certificate export outside AWS. Microsoft Azure's Key Vault allows for centralized management of certificates, including those from external CAs, and integrates seamlessly with Azure services. Google Cloud provides managed SSL certificates specifically for its load balancers, offering automatic renewals and deployment.

Additionally, for developers and small businesses that need a quick, cost-effective solution, automated CAs such as Let's Encrypt offer **Domain Validated (DV)** certificates at no cost. These certificates are issued using automated processes, designed to ensure that the applicant controls the domain listed in the certificate. This method is highly efficient and perfect for securing websites rapidly.

Ultimately, the choice of a certificate authority depends on our specific security needs, the level of trust we require, and our budget. Each type of CA offers different features and levels of security, making it crucial to select one that aligns with your objectives for security and customer trust.

## See also

Read more about SSL and TLS at <https://www.secdops.com/blog/ssl-tls-and-https-a-beginners-guide-to-web-security>.

## Creating an SSL/TLS certificate with ACM

In this recipe, we will create an **X.509 certificate** for a public domain that we own using **AWS Certificate Manager (ACM)**. ACM public certificates are used with AWS services such as **Elastic Load Balancing (ELB)**, **Amazon CloudFront**, **AWS Elastic Beanstalk**, **Amazon API Gateway**, and **AWS CloudFormation**.

## Getting ready

We need the following to successfully complete this recipe:

- A working AWS account, `awssecb-sandbox-1`, and a user, `awssecbadmin1`, as described in the *Technical requirements* section.
- A domain name with any domain name provider (including AWS), with access to its control panel. I will be using a domain with the name `trainso.io`.

## How to do it...

We can create a TLS certificate in ACM as follows:

1. Go to the **AWS Certificate Manager** dashboard. If you're using ACM for the first time, you should see the **Get started** options. Currently, AWS provides options so that we can provision certificates, as well as create a private CA.

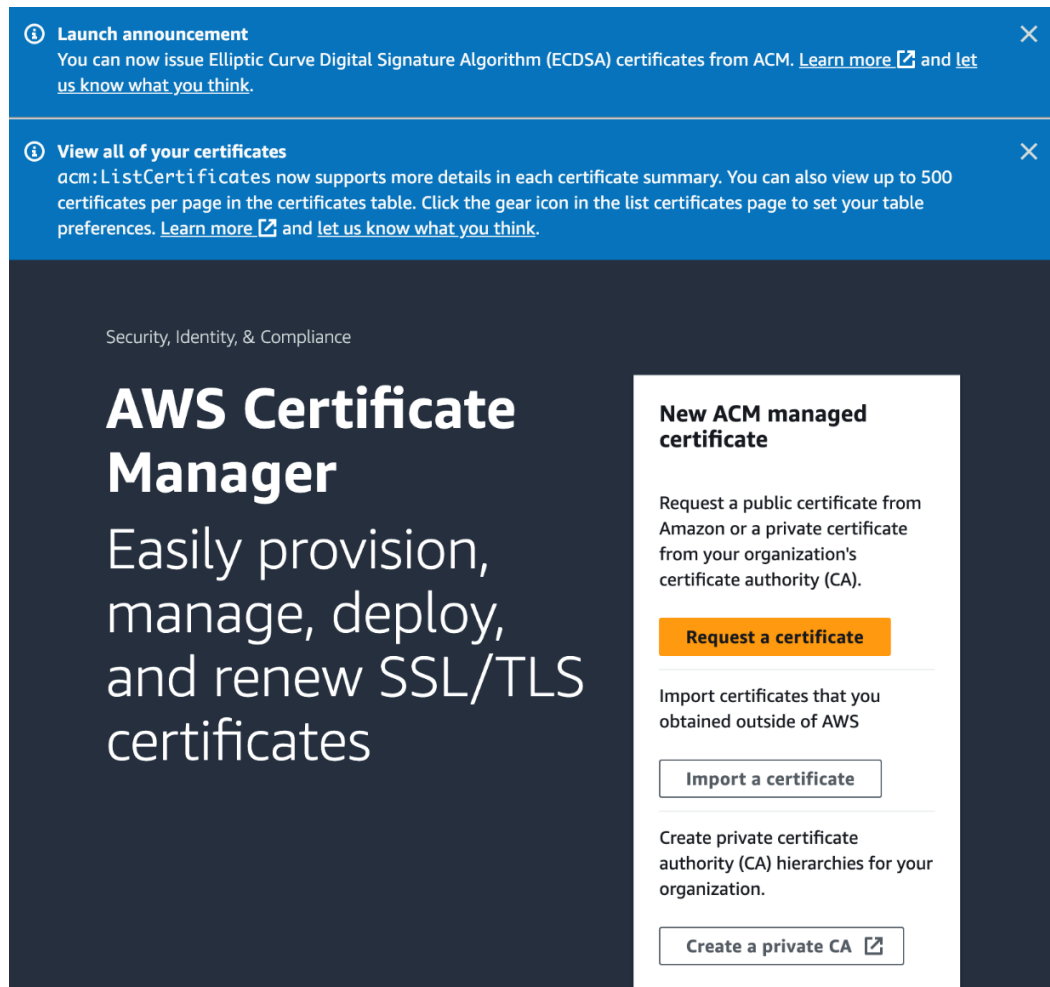


Figure 6.4 – The AWS Certificate Manager dashboard

2. In the left sidebar, we should see the **Import a certificate**, **List certificates**, **Request a certificate**, and **AWS Private CA** options. Click on **Request a certificate**.



3. Select **Request a public certificate** and click on **Next**.

AWS Certificate Manager > Certificates > Request certificate

## Request certificate

**Certificate type** [Info](#)

ACM certificates can be used to establish secure communications access across the internet or within an internal network. Choose the type of certificate for ACM to provide.

☒ **Request a public certificate**  
Request a public SSL/TLS certificate from Amazon. By default, public certificates are trusted by browsers and operating systems.

☐ **Request a private certificate**  
Requesting a private certificate requires the creation of a private certificate authority (CA). To create a private CA, visit [AWS Private Certificate Authority](#) [↗](#)

**Cancel** **Next**

Figure 6.5 – Certificate type configuration

4. Enter a fully qualified domain name in the **Domain names** textbox. To include all sub-domains, I will use a wildcard with the domain name `*.trainso.io`.
5. For **Validation method**, select **DNS validation - recommended**.

**Validation method** [Info](#)

Select a method for validating domain ownership.

☒ **DNS validation - recommended**  
Choose this option if you are authorized to modify the DNS configuration for the domains in your certificate request.

☐ **Email validation**  
Choose this option if you do not have permission or cannot obtain permission to modify the DNS configuration for the domains in your certificate request.

Figure 6.6 – Selecting a validation method

6. For **Key algorithm**, select **RSA 2048**.

### Key algorithm [Info](#)

Select an encryption algorithm. Some algorithms may not be supported by all AWS services.

- ☒ **RSA 2048**  
RSA is the most widely used key type.
- ☐ **ECDSA P 256**  
Equivalent in cryptographic strength to RSA 3072.
- ☐ **ECDSA P 384**  
Equivalent in cryptographic strength to RSA 7680.

Figure 6.7 – Selecting key algorithm

7. On the **Add Tags** screen, add tags if required and click on **Request**. We should see the new certificate on the **Certificates** page. We can also reach the **Certificates** page using the **List certificates** option from the left sidebar. The status of our certificate will be **Pending validation**.
8. On the **Certificates** page, click on the hyperlinked certificate ID to go to the certificate's page.
9. Scroll down to the **Domains** section, copy the CNAME name and CNAME value, and add them to our domain's DNS records from the domain provider's control panel. Note that with most domain providers, you will not have to copy and paste the whole CNAME (short for Canonical Name) value but only the part before the first dot (.). For example, if our CNAME value is `_b262683f801a4eb13d8eb4a36cd8a2ba.trainso.io.`, we may have to only enter `_b262683f801a4eb13d8eb4a36cd8a2ba` as the CNAME value in our domain provider's control panel, excluding the trailing `.trainso.io.`. The domain provider could be Amazon Route 53 or an external provider, such as Namecheap or GoDaddy. Configuring DNS is outside the scope of this book, but I will provide some useful references in the *See also* section of this recipe.
10. Once the CNAME record is updated, we can go to the **Certificates** screen and check the status. Once successful, the value of the **Status** column of our certificate will change to **Issued**. It could take some time for the DNS changes to propagate. If the status is **Pending validation**, check again after some time, or refresh the page using the refresh button, until the status has changed to **Issued**.

## How it works...

In this recipe, we created a certificate using ACM. We can request a certificate for one or more domain names. We need to specify a fully qualified domain name, such as `www.trainso.io`, or one that uses a wildcard, such as `*.trainso.io`, representing all the subdomains of `trainso.io`.

Before issuing the certificate, we need to validate the ownership of the domain. We can do this either through DNS validation or email validation. We performed DNS validation in this recipe. AWS will provide a CNAME record per domain for DNS validation. We need to update this CNAME record on our domain's DNS management service. Route 53 is Amazon's DNS management service.

## There's more...

ACM public certificates are supported for AWS services such as ELB, Amazon CloudFront, AWS Elastic Beanstalk, Amazon API Gateway, and AWS CloudFormation. AWS does not allow us to use the ACM public certificates to enable SSL/TLS on our EC2 instances. However, ACM private CA-issued certificates can be used with EC2 instances, containers, and even our own servers.

AWS does not charge us for the public TLS certificates that are provisioned through ACM. We only need to pay for the AWS resources we create to run our application. However, creating an ACM private CA is not free. For a private CA, we are charged a monthly fee, as well as for the private certificates we issue. We are not charged once we delete a private CA; however, if we restore a private CA, we will be charged for the time it was in the deleted state.

## See also

- We can learn more about DNS here: <https://www.secdops.com/blog/mastering-dns-resources-and-recipes>.
- We can read more about ACM service integrations and view the current list of supported services here: <https://docs.aws.amazon.com/acm/latest/userguide/acm-services.html>.

## Creating ELB target groups

In this recipe, we will learn how to create an **Elastic Load Balancer (ELB) target group**. The application load balancer and network load balancer route traffic to ELB target groups, unlike classic load balancers, which route traffic to individual EC2 instances.

## Getting ready

To follow this recipe, we need the following:

- A working AWS account, `awssecb-sandbox-1`, and a user `awssecbadmin1`, as described in the *Technical requirements* section.
- A VPC, `awssecb-vpc`, following the *Setting up VPC plus VPC resources with minimal efforts* recipe from *Chapter 5*.

- Two EC2 instances in the aforementioned `awssecceb-vpc` VPC, following the *Launching an EC2 instance with a web server using user data* recipe in *Chapter 5*, with the following exceptions. Give names as `Cloudericks Web Server` and `Cloudericks Web Server 2`, respectively, while creation.

**Tip**

Name is internally represented using a tag with key set to Name.

- Subnets should be public and selected in the `us-east-1a` and `us-east-1b` availability zones, respectively. For the second instance, replace `Cloudericks Web Server` within the user data with `Cloudericks Web Server 2` to distinguish the two web servers. The key pair and security group can be shared between the two instances.
- Before proceeding, make sure that our instances are running and accessible directly from a browser. For additional security, after the ELB has been configured and tested, we can restrict access to these instances from our ELB's security group.

## How to do it...

We can create a target group as follows:


1. Log into the AWS Management Console and go to the **EC2** service.
2. In the left sidebar, under **Load Balancing**, click on **Target Groups**.
3. Click **Create target group**.
4. On the **Specify group details** page, under the **Basic configuration** section, select **Instances** for **Choose a target type**.

## Basic configuration

Settings in this section can't be changed after the target group is created.

### Choose a target type

☒ **Instances**

- Supports load balancing to instances within a specific VPC.
- Facilitates the use of [Amazon EC2 Auto Scaling](#)  to manage and scale your EC2 capacity.

☐ **IP addresses**

- Supports load balancing to VPC and on-premises resources.
- Facilitates routing to multiple IP addresses and network interfaces on the same instance.
- Offers flexibility with microservice based architectures, simplifying inter-application communication.
- Supports IPv6 targets, enabling end-to-end IPv6 communication, and IPv4-to-IPv6 NAT.

☐ **Lambda function**

- Facilitates routing to a single Lambda function.
- Accessible to Application Load Balancers only.

☐ **Application Load Balancer**

- Offers the flexibility for a Network Load Balancer to accept and route TCP requests within a specific VPC.
- Facilitates using static IP addresses and PrivateLink with an Application Load Balancer.

Figure 6.8 – Choosing a target type

5. For **Target group name**, enter `cloudericks-tg`, or a meaningful name specific to your user case.
6. For **Protocol: Port**, select **HTTP**; for **Port**, enter 80; and for **IP address type**, select **IPv4**.

7. For **VPC**, select the `awssecb-vpc` VPC, where our EC2 instances are present.
8. Under **Protocol version**, select **HTTP1**.

Protocol version

- ☒ **HTTP1**  
Send requests to targets using HTTP/1.1. Supported when the request protocol is HTTP/1.1 or HTTP/2.
- ☐ **HTTP2**  
Send requests to targets using HTTP/2. Supported when the request protocol is HTTP/2 or gRPC, but gRPC-specific features are not available.
- ☐ **gRPC**  
Send requests to targets using gRPC. Supported when the request protocol is gRPC.

Figure 6.9 – Selecting the protocol version

9. In the **Health checks** section, for **Health check protocol**, select **HTTP**, and for **Health check path**, enter `/index.html`. Leave other values as-is and click on **Next**.
10. In the **Register targets** pane, select the `Cloudericks Web Server` and `Cloudericks Web Server 2` EC2 instances and click on **Include as pending below**. Instances should now appear under **Review targets**.
11. Click on **Create target group**. We should see a message that the target group has been created successfully.

The health status of the targets will now be **Unused**. It will change after they have been attached to an ELB.

## How it works...

In this recipe, we created a target group for EC2 instances with the HTTP protocol. We can create an application load balancer with a target group, using the HTTP or HTTPS protocol. A network load balancer needs a target group with a TCP or TLS protocol. We can also create target groups for IP addresses and AWS Lambda functions. By selecting the **IP addresses** option, we can select public IP addresses that are outside of AWS.

For the health check, we set the protocol to HTTP and the path to `/index.html`. We can override the port for a health check, if necessary, by selecting the **Override** option for the port, under **Advanced health check settings**. We can set the time to wait for a response from an instance (timeout), the time between health checks (interval), the number of consecutive failures before declaring an instance unhealthy (unhealthy threshold), the number of consecutive successes before declaring an instance healthy (healthy threshold), and the HTTP response codes to check for success (success codes).

The target group instances will have an initial state of unused when they are created. When the target group is attached to an ELB, the status will change to initial. If the health checks pass, then the status changes to **Healthy**. Other supported statuses include unhealthy if the health check fails, or draining if the target is being unregistered and connection draining is happening.

## There's more...

In this recipe, we created a target group with the HTTP protocol. Target groups can be created with the HTTP, HTTPS, TCP, TLS, UDP, GENEVE, and TCP\_UDP protocols. We can follow the steps in this recipe to create target groups with other protocols. For example, we can add a target group with the HTTPS protocol and the port set to 443, and then add EC2 instances with SSL/TLS enabled to the target group. For TCP passthrough of an HTTPS request with a network load balancer, which is needed for TLS termination at EC2, we should set the protocol to TCP but with the port set to 443.

## See also

We can read more about load balancing in AWS at <https://www.cloudericks.com/blog/understanding-load-balancing-in-aws>.

# Using an application load balancer with TLS termination at the ELB

**Application load balancers (ALBs)** work at the request layer (application layer of the OSI model) and are used for HTTP and HTTPS requests. ALBs provide advanced routing capabilities at the application layer for requesting and path parameter-based routing. Architecture patterns, such as microservices architecture, can use ALBs to route requests to different web servers while making use of request parameters.

## Getting ready

To follow this recipe, we need the following:

- A working AWS account, `awssecbb-sandbox-1`, and a user `awssecbbadmin1`, as described in the *Technical requirements* section.
- Create a target group, `cloudericks-tg`, with two EC2 instances, following the *Creating ELB target groups* recipe of this chapter, and the following resources should have been created as part of the preparation for that recipe – a VPC, `awssecbb-vpc`, and a security group, `cloudericks-web-server`.
- To select HTTPS (secure HTTP) as the ELB listening protocol, we need an ACM certificate. We can create an ACM certificate by following the *Creating an SSL/TLS certificate with ACM* recipe of this chapter.

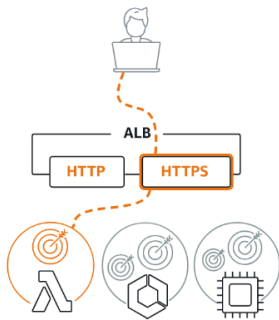
## How to do it...

We can create and test an ALB as follows:

1. Go to the **EC2** service in the console.
2. In the left sidebar, under **Load Balancing**, click on **Load Balancers** from the left sidebar.
3. Click on **Create load balancer**.
4. We should see the options to create the three primary types of load balancers – namely, **Application Load Balancer**, **Network Load Balancer**, and **Gateway Load Balancer**. Click **Create** under **Application Load Balancer**.

### Load balancer types

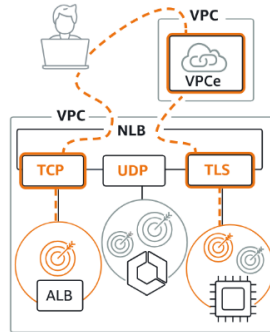
#### Application Load Balancer [Info](#)



Choose an Application Load Balancer when you need a flexible feature set for your applications with HTTP and HTTPS traffic. Operating at the request level, Application Load Balancers provide advanced routing and visibility features targeted at application architectures, including microservices and containers.

Create

#### Network Load Balancer [Info](#)



Choose a Network Load Balancer when you need ultra-high performance, TLS offloading at scale, centralized certificate deployment, support for UDP, and static IP addresses for your applications. Operating at the connection level, Network Load Balancers are capable of handling millions of requests per second securely while maintaining ultra-low latencies.

Create

#### Gateway Load Balancer [Info](#)



Choose a Gateway Load Balancer when you need to deploy and manage a fleet of third-party virtual appliances that support GENEVE. These appliances enable you to improve security, compliance, and policy controls.

Create

► **Classic Load Balancer** - previous generation

Figure 6.10 – The load balancer types



5. On the **Basic configuration** screen, for **Load balancer name**, enter `cloudericks-app-lb`, or a meaningful name as required. For **Scheme**, select **Internet-facing**, and for **IP address type**, select **IPv4**.

## Basic configuration

### Load balancer name

Name must be unique within your AWS account and can't be changed after the load balancer is created.


cloudericks-app-lb

A maximum of 32 alphanumeric characters including hyphens are allowed, but the name must not begin or end with a hyphen.

### Scheme [Info](#)

Scheme can't be changed after the load balancer is created.

☒ **Internet-facing**

An internet-facing load balancer routes requests from clients over the internet to targets. Requires a public subnet. [Learn more](#) 

☐ **Internal**

An internal load balancer routes requests from clients to targets using private IP addresses.

### IP address type [Info](#)

Select the type of IP addresses that your subnets use.

☒ **IPv4**

Includes only IPv4 addresses.

☐ **Dualstack**

Includes IPv4 and IPv6 addresses.

Figure 6.11 – Basic configuration

6. In the **Network mapping** section, for **VPC**, select our VPC, `awsseccb-vpc`.
7. In the **Network mapping** section, under **Mappings**, select the availability zones and public subnets where we have our instances.

**Mappings** [Info](#)

Select at least two Availability Zones and one subnet per zone. The load balancer routes traffic to targets in these Availability Zones only. Availability Zones that are not supported by the load balancer or the VPC are not available for selection.

☒ **us-east-1a (use1-az6)**

Subnet

subnet-0207b1bee6e911ac9

awssecb-subnet-public1-us-east-1a ▼

IPv4 address

Assigned by AWS

☒ **us-east-1b (use1-az1)**

Subnet

subnet-004c1d6e5f5f37b57

awssecb-subnet-public2-us-east-1b ▼

IPv4 address

Assigned by AWS

Figure 6.12 – Selecting the availability zones and subnets

8. Select the `cloudericks-web-server` security group, which allows **HTTP** and **HTTPS** from **Anywhere**.
9. Under **Listeners and routing**, set **Protocol** to **HTTPS**. The port will be set automatically to 443. For **Default action**, select our target group, `cloudericks-tg`.

▼ Listener **HTTPS:443**

Remove

Protocol

Port

Default action

[Info](#)

HTTPS ▼

:

443

Forward to

cloudericks-tg

HTTP ▼

1-65535

Target type: Instance, IPv4

↻

[Create target group](#) [↗](#)

Figure 6.13 – Adding an HTTPS Listener

- Under **Secure listener settings**, for **Certificate source**, select **From ACM**, and for **Certificate name**, select the ACM certificate we created for this recipe.

### Default SSL/TLS server certificate

The certificate used if a client connects without SNI protocol, or if there are no matching certificates. You can source this certificate from AWS Certificate Manager (ACM), Amazon Identity and Access Management (IAM), or import a certificate. This certificate will automatically be added to your listener certificate list.

#### Certificate source



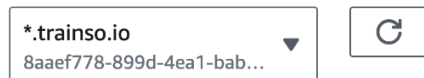
☒ From ACM

☐ From IAM

☐ Import certificate

#### Certificate (from ACM)

The selected certificate will be applied as the default SSL/TLS server certificate for this load balancer's secure listeners.



\*.trainso.io ▼

8aaef778-899d-4ea1-bab...

↻

[Request new ACM certificate](#) 

Figure 6.14 – Selecting an SSL/TLS server certificate

- Optionally, you can include an AWS WAF or create an **AWS Global Accelerator** for this load balancer, as shown in the following screenshot. Let's skip them for now.

### Optimize with service integrations - *optional*

Optimize your load balancing architecture by integrating AWS services with this load balancer at launch. You can also add these and other services after your load balancer is created by reviewing the load balancer's "Integrations" tab.



#### AWS Web Application Firewall (WAF) [Info](#)

[Additional charges apply](#)

Optimizes: **Security**

☐ **Include WAF security protections behind the load balancer**

Associates a pre-defined web ACL that includes the AWS-recommended security protections. Alternatively, you can associate any of your existing WAF web ACLs for custom protections.



#### AWS Global Accelerator [Info](#)

[Additional charges apply](#)

Optimizes: **Performance, Availability, Security**

☐ **Create an accelerator**

An accelerator will be created in your account. The accelerator provides 2 global static IPs that act as a fixed entry point to your load balancer.

Figure 6.15 – Configuring AWS WAF and AWS Global Accelerator

12. Review the details and click **Create load balancer**. If we go to the target group, cloudericks-tg, the health status of our instances will be first **Initial**, and after some time, the status should change to **Healthy**.
13. Copy the DNS name from the **Description** tab of the ELB, add the `https://` prefix to it, as shown in the following figure, and enter this URL in a browser. Should a security warning appear, select **Advanced**, and then click the link to continue. This will display one of the web servers. By refreshing the page multiple times, you can observe the response from both web servers.



## Cloudericks Web Server 2

Figure 6.16 – The response from the load balancer, showing our first instance data

We will get a warning, since our URL (ELB DNS) does not match the certificate's domain, which is `*.trainso.io` in my case.

14. Create a **CNAME** record for the domain, with **Name** (or **Host**) set to `cloudericks.trainso.io` (this will be only `cloudericks` if the DNS service provider automatically appends the domain name) and **Value** as our DNS name, which in my case is `cloudericks-app-lb-901703641.us-east-1.elb.amazonaws.com`. Remember to replace `trainso.io` with the domain for which you created the certificate.
15. Once the DNS changes are propagated, which could take some time, we should be able to run our subdomain URL (e.g., `cloudericks.trainso.io` in my case) and we should get a successful response, as seen in the following figure:

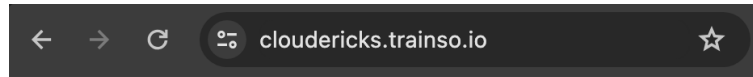


Figure 6.17 – A successful HTTPS response with an SSL/TLS certificate

#### Important note

There are multiple ways to point our domain with ELB, including creating a Route 53 account and changing the names of the servers of our domain, or adding a CNAME record for a subdomain.

## How it works...

In this recipe, we created an internet-facing load balancer. We set the listening protocol to HTTPS (secure HTTP), and on the **Configure Security Settings** page, we selected an ACM certificate. We set the security policy to `ELBSecurityPolicy-2016-08`. The security policy is an SSL negotiation configuration that's used to negotiate SSL connections with clients.

We terminated TLS at the ELB level. Note that the connection to the instance from the ELB is without TLS. The ALB only supports TLS/SSL termination at the ELB level. The network load balancer and classic load balancer can be used to terminate TLS/SSL at the EC2 instance level, by using the TCP protocol on port 443.

When we terminate TLS at the ELB for an HTTPS request, the request is decrypted at the ELB and sent unencrypted to the EC2 instances over the private network within our VPC. When we terminate an HTTPS request's TLS at the EC2 instance, the request is not decrypted at the ELB and is only decrypted at the EC2 instance.

Terminating TLS at the ELB level avoids the overhead of TLS termination at EC2 instances and is more efficient. However, if there is a compliance requirement for end-to-end encryption, we should terminate at the EC2 instance level.

## There's more...

Let's quickly go through some more important concepts related to ALBs:

- ALBs only support HTTP or HTTPS protocols. For other protocols, such as TCP, we need to use a network load balancer or classic load balancer.
- ALBs only support TLS/SSL termination at the ELB level.
- We can enable sticky sessions for ALBs at the target group level. However, we cannot enable sticky sessions at individual EC2 instances with ALBs.
- We can do path-based routing with ALBs if path patterns are enabled.
- We set the security policy for SSL/TLS negotiation to `ELBSecurityPolicy-2016-08`, which is the default. The following are the currently available policies: `ELBSecurityPolicy-2016-08`, `ELBSecurityPolicyTLS-1-2-2017-01`, `ELBSecurityPolicy-TLS-1-1-2017-01`, `ELBSecurityPolicyTLS-1-2-Ext-2018-06`, `ELBSecurityPolicy-FS-2018-06`, `ELBSecurityPolicy-2015-05`, `ELBSecurityPolicy-TLS-1-0-2015-04`, `ELBSecurityPolicy-FS-1-2-Res-2019-08`, `ELBSecurityPolicy-FS-1-1-2019-08`, and `ELBSecurityPolicy-FS-1-2-2019-08`.

## See also

We can read more about creating a listener for our ALB at <https://docs.aws.amazon.com/elasticloadbalancing/latest/application/create-https-listener.html>.

## Using a network load balancer with TLS termination at EC2

**Network load balancers** are used to load-balance TCP traffic and work at layer 4 of the OSI model. They provide very high performance compared to other load balancer types and can support millions of requests per second, with very low latencies.

## Getting ready

To follow this recipe, we need the following:

- A working AWS account, `awsseccb-sandbox-1`, and a user `awsseccbadmin1`, as described in the *Technical requirements* section.

- Create a target group, `cloudericks-tg-tcp`, following the *Creating ELB target groups* recipe of this chapter, but selecting TCP instead of HTTP as the protocol and the port as 443. The following resources should have been created as part of the preparation for that recipe – a VPC, `awssecb-vpc`, and a security group, `cloudericks-web-server`. For this recipe, we need an EC2 instance, following the *Launching an EC2 Instance with a web server using user data* recipe from *Chapter 5*.
- Enable HTTPS for our web server, following the *Enabling HTTPS for a web server on an EC2 instance* recipe, as we want HTTPS on an EC2 instance for this recipe. Make sure that our instances can be reached using the `https://` prefix.
- To select HTTPS (secure HTTP) as the ELB listening protocol, we need an ACM certificate. We can create an ACM certificate by following the *Creating an SSL/TLS certificate with ACM* recipe of this chapter.

## How to do it...

We can create and test a network load balancer with TLS termination at an EC2 instance, as follows:

1. Log into the AWS Management Console and go to the **EC2** service.
2. Click on **Load Balancers** from the left sidebar.
3. Click on **Create load balancer**. We should see the options to create three types of load balancers, **Application Load Balancer**, **Network Load Balancer**, and **Classic Load Balancer**, as shown in *Figure 6.10*.
4. Under **Network Load Balancer**, click on **Create**.
5. On the **Create Network Load Balancer** screen, under the **Basic configuration** section, enter `cloudericks-nw-lb` for **Load balancer name**; for **Scheme**, select **internet-facing**; and for **IP address type**, select **IPv4**.
6. In the **Network mapping** section, for **VPC**, select our VPC, `awssecb-vpc`.
7. In the **Network mapping** section, under **Mappings**, select the availability zones and public subnets where we have our instances, as shown in *Figure 6.12*.
8. Under **Security groups**, select the `cloudericks-web-server` security group.
9. For **Protocol**, select **TCP**, and set the value for **Port** to 443. For **Target group**, select `cloudericks-tg-tcp`.
10. Optionally, you can create an AWS Global Accelerator for this load balancer. Let's skip it for now. With the ALB, we also had the option to include an AWS WAF, as we saw in *Figure 6.15*.
11. Leave other details as-is and click on **Create load balancer**.

12. Copy the DNS name from the **Description** tab of the ELB, add the `https://` prefix to it, as shown in the following figure, and enter this URL in a browser. Should a security warning appear, select **Advanced**, and then click the link to continue. This will display one of the web servers. By refreshing the page multiple times, you can observe the response from both web servers.



## Cloudericks Web Server 2

Figure 6.18 – The network load balancer response

Here, we are performing a TLS termination in the EC2 instance; hence, we do not have to configure a certificate for the load balancer. We get a warning in the browser because we are using a self-signed certificate. If we use a certificate signed by a CA, we will not receive a warning.

### How it works...

In this recipe, we created a **network load balancer (NLB)** with TLS termination at EC2. Most of the options were the same as what we saw in the *Using an application load balancer with TLS termination at the ELB* recipe of this chapter. In this recipe, we used the TCP protocol and port 443. This was done to allow the NLB to simply pass the HTTPS request to the EC2 instance without decrypting it at the ELB level. The target group should also be configured with the TCP protocol and with port 443, allowing TCP passthrough. If we select TLS (secure TCP) instead of TCP, NLB will decrypt the request at ELB itself.

### There's more...

In this recipe, we did TCP passthrough for an HTTPS request and performed TLS termination at the EC2 instance. TLS termination at the EC2 instance will consume more EC2 resources and provide an extra load for the EC2 instance. We will also need to manage the certificate across all the EC2 instances. However, if we require end-to-end encryption due to compliance or government policies, this is the preferred way. Otherwise, the preferred approach is to perform SSL/TLS termination at the ELB level, as we saw in the *Using an application load balancer with TLS termination at the ELB* recipe of this chapter. To terminate SSL/TLS at the NLB, we need to set the protocol to TLS (secure TCP) and select an ACM certificate.

### See also

We can read more about TLS termination for load balancers in AWS here: <https://www.cloudericks.com/blog/understanding-tls-termination-with-load-balancers-in-aws>.



## Securing S3 using CloudFront and TLS

In this recipe, we will learn how to secure an S3 bucket by adding a CloudFront distribution layer. We will enable SSL/TLS on the CloudFront distribution to allow HTTPS traffic. Initially, we will utilize the default CloudFront certificate (`*.cloudfront.net`), and then proceed to configure the CloudFront distribution with a custom domain, using an ACM certificate.

### Getting ready

We need the following to successfully complete this recipe.

- A working AWS account, `awssecb-sandbox-1`, and a user `awssecbadmin1`, as described in the *Technical requirements* section.
- We need an S3 bucket with a file called `index.html`. The content of the file should be `<h1>Cloudericks Web Server </h1>`. We can create an S3 bucket by referring to *Technical requirements* section of *Chapter 2*.
- For the *CloudFront distribution with a custom domain and ACM certificate* section of this recipe, we need to create an ACM certificate, following the *Creating an SSL/TLS certificate with an ACM* recipe of this chapter.

### How to do it...

We can add a CloudFront distribution to an S3 bucket with or without a custom domain. We will see both approaches in this recipe.

#### *CloudFront distribution with a default CloudFront domain*

We can add a CloudFront distribution to an S3 bucket with a default CloudFront domain and certificate as follows:

1. Log into the AWS Management Console and go to the **CloudFront** service.
2. If you are new to CloudFront, you should see a screen with a **Create a CloudFront distribution** button. Click on it. Otherwise, you can first click on **Distributions** from the left sidebar and then **CloudFront distribution**.

Amazon CloudFront is a fast content delivery network (CDN) service that securely delivers data, videos, applications, and APIs to customers globally with low latency and high transfer speeds.

### Get started with CloudFront

Enable accelerated, reliable and secure content delivery for Amazon S3 buckets, Application Load Balancers, Amazon API Gateway APIs, and more in 5 minutes or less.

**Create a CloudFront distribution**

Figure 6.19 – Creating a CloudFront distribution

3. In the **Origin** pane, For **Origin domain**, select the S3 bucket we created for this recipe. Leave **Origin Path** empty. For **Name**, use the auto-generated value. For **Origin access**, select **Origin access control settings**.

#### Origin domain

Choose an AWS origin, or enter your origin's domain name.

cloudericks-web-server.s3.us-east-1.amazonaws.com

#### Origin path - optional

Enter a URL path to append to the origin domain name for origin requests.

Enter the origin path

#### Name

Enter a name for this origin.

cloudericks-web-server.s3.us-east-1.amazonaws.com

#### Origin access [Info](#)

☐ Public

Bucket must allow public access.

☒ Origin access control settings (recommended)

Bucket can restrict access to only CloudFront.

☐ Legacy access identities

Use a CloudFront origin access identity (OAI) to access the S3 bucket.

#### Origin access control

Select an existing origin access control (recommended) or create a new control.

Select an origin access control

Create new OAC

Figure 6.20 – Setting the origin details

- Click on **Create new OAC**. For the name, we can use the auto-populated name. For the signing behavior, select **Sign requests (recommended)**. Leave everything as default, as shown in the following figure, and click on **Create**.

**Create new OAC** ✕

**Name**  
The name must be unique. Valid characters: letters, numbers and most special characters. Use up to 64 characters.

**Description - optional**  
The description can have up to 256 characters.

**Signing behavior**  
☐ Do not sign requests  
☒ **Sign requests (recommended)**  

☐ Do not override authorization header  
Do not sign if incoming request has authorization header.

**Origin type**  

S3 ▼

  
The origin type must be the same type as origin domain.

Cancel

Create

Figure 6.21 – Creating a new OAC

- Select the new OAC on our **Create distribution** page. Leave the value selection for **Enable Origin Shield** as **No**.

- Under **Default cache behavior**, for **Path pattern**, use the default value; for **Compress objects automatically**, select **Yes**; for **Viewer protocol policy**, select **Redirect HTTP to HTTPS**; for **Allowed HTTP methods**, select **GET, HEAD**; and for **Restrict viewer access**, select **No**.

### Default cache behavior

Path pattern [Info](#)

Default (\*)

Compress objects automatically [Info](#)

☐ No

☒ Yes

### Viewer

Viewer protocol policy

☐ HTTP and HTTPS

☒ Redirect HTTP to HTTPS

☐ HTTPS only

Allowed HTTP methods

☒ GET, HEAD

☐ GET, HEAD, OPTIONS

☐ GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE

Restrict viewer access

If you restrict viewer access, viewers must use CloudFront signed URLs or signed cookies to access your content.

☒ No

☐ Yes

Figure 6.22 – Configuring the default cache behavior

- Leave the default settings for **Cache key and origin requests** and **Function associations**.

8. Scroll down to **Web Application Firewall (WAF)** and select the **Do not enable security protections** option.

### Web Application Firewall (WAF) [Info](#)

☐ **Enable security protections**

Keep your application secure from the most common web threats and security vulnerabilities using AWS WAF. Blocked requests are stopped before they reach your web servers.

☒ **Do not enable security protections**

Select this option if your application does not need security protections from AWS WAF.

Figure 6.23 – The WAF selection options

9. For **Price class**, select **Use all edge locations (best performance)**.

### Settings

**Price class** [Info](#)

Choose the price class associated with the maximum price that you want to pay.

☒ **Use all edge locations (best performance)**

☐ Use only North America and Europe

☐ Use North America, Europe, Asia, Middle East, and Africa

**Alternate domain name (CNAME) - optional**

Add the custom domain names that you use in URLs for the files served by this distribution.

**Add item**

 To add a list of alternative domain names, use the [bulk editor](#).

Figure 6.24 – Selecting the price class

- For **Custom SSL certificate - optional**, do not make any selections; for **Supported HTTP versions**, select **HTTP/2**; and for **Default root object - optional**, enter `index.html`.

#### Custom SSL certificate - optional

Associate a certificate from AWS Certificate Manager. The certificate must be in the US East (N. Virginia) Region (us-east-1).

Choose certificate ▼



[Request certificate](#)

#### Supported HTTP versions

Add support for additional HTTP versions. HTTP/1.0 and HTTP/1.1 are supported by default.

☒ HTTP/2

☐ HTTP/3

#### Default root object - optional

The object (file name) to return when a viewer requests the root URL (/) instead of a specific object.

index.html

Figure 6.25 – Selecting the SSL certificate and default root object

- For **Standard logging**, select **Off**, and for **IPv6**, select **On**. Standard logging, when enabled, retrieves logs of viewer requests, and delivers them to an Amazon S3 bucket. Consequently, enabling this feature requires specifying the target bucket and determining whether cookie logging is needed. With cookie logging enabled, CloudFront includes cookies in the standard logs.
- Click **Create distribution**. We will get a prompt with the option to copy the S3 bucket policy that needs to be updated in our S3 bucket.

✓ Successfully created new distribution.

#### ⚠ The S3 bucket policy needs to be updated

Complete distribution configuration by allowing read access to CloudFront origin access control in your policy statement. [Go to S3 bucket permissions to update policy](#)

Copy policy

Figure 6.26 – The copy S3 policy notification

13. Click on **Copy policy** and paste this policy into the **Bucket policy** section of our S3 bucket. If you are not sure how to do this, you can do it by referring to the *Creating an S3 bucket policy* recipe in *Chapter 4*. If you miss this step, you will need to set it up independently by consulting the Amazon documentation at <https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/private-content-restricting-access-to-s3.html>.
14. Copy the **Distribution domain name** from our CloudFront distribution and run it from a browser. We should get an output like the following screenshot:

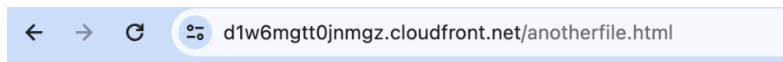


## Cloudericks Web Server

Figure 6.27 – A response through the CloudFront Distribution

Here, we did not have to specify a filename, as we had specified a default object.

15. Optionally, we can also specify a filename within our S3 bucket directly, as shown in the following screenshot. This file was not created in the *Getting ready* section, so you will need to create it if you want to run a similar URL.



## Cloudericks Web Server - Another File!

Figure 6.28 – Directly accessing a file in S3

Next, we will learn how to use a custom domain with our own ACM certificate.

### *CloudFront distribution with a custom domain and ACM certificate*

We can add a CloudFront distribution to an S3 bucket with a custom domain and ACM certificate, as follows:

1. Follow *Steps 1 to 9* in the *CloudFront distribution with a default CloudFront domain* section of this recipe, up to setting the price class. However, there is one exception – if an **Origin Access Control (OAC)** was created in that section, there is no need to create a new one; we can reuse the existing OAC.

2. For **Alternate domain name (CNAME) - optional**, enter a subdomain name for the domain for which we have the certificate, as shown in the following screenshot. Remember to replace `trainso.io` with your domain name.

#### Alternate domain name (CNAME) - optional

Add the custom domain names that you use in URLs for the files served by this distribution.

Figure 6.29 – Setting the alternate domain name (CNAME)

3. For **Custom SSL certificate**, as shown in *Figure 6.30*, select the ACM certificate we created for this recipe, uncheck **Legacy clients support**, and for **Security policy**, select **TLSv1.2\_2021**, which is the current recommended version. For **Supported HTTP versions**, select **HTTP/2**, and for **Default root object - optional**, enter `index.html`.

#### Custom SSL certificate - optional

Associate a certificate from AWS Certificate Manager. The certificate must be in the US East (N. Virginia) Region (us-east-1).

☒ \*.trainso.io [Request certificate](#)

Legacy clients support - \$600/month prorated charge applies. Most customers do not need this.

CloudFront allocates dedicated IP addresses at each CloudFront edge location to serve your content over HTTPS.

☐ Enabled

#### Security policy

The security policy determines the SSL or TLS protocol and the specific ciphers that CloudFront uses for HTTPS connections with viewers (clients).

- ☒ TLSv1.2\_2021 (recommended)
- ☐ TLSv1.2\_2019
- ☐ TLSv1.2\_2018
- ☐ TLSv1.1\_2016
- ☐ TLSv1\_2016
- ☐ TLSv1

Figure 6.30 – ACM certificate configuration

4. For **Standard logging**, select **Off**, and for **IPv6**, select **On**.



5. Click **Create distribution**. The copy policy message will be shown with a bucket policy. Ensure to update the bucket policy of our S3 bucket with this policy, as we did in the previous section.
6. Create a **CNAME** record for the domain, with the name (or host) set to the subdomain we entered in *Step 2*, which was `cloudericksws.trainso.io` in my case (this is only `cloudericksws` if the DNS service provider automatically appends the domain name) and the value set as our CloudFront domain name, which in my case is `d31z3ldyzun0k3.cloudfront.net`.
7. After some time, considering the DNS propagation delay, run the subdomain from a browser, and we should get a successful response.



Figure 6.31 – A response using the custom domain

Instead of the custom subdomain, we can also reach the web page using the CloudFront domain name, as we saw in the previous section.



Figure 6.32 – A response using the CloudFront domain name

In this section, we accessed the content of our private S3 bucket through CloudFront, using a custom domain and an ACM certificate.

## How it works...

In this recipe, we created a CloudFront distribution layer over a private S3 bucket to access the S3 bucket securely using HTTPS. We configured this to redirect all HTTP requests to HTTPS requests. If we choose HTTP and HTTPS, both HTTP and HTTPS requests are allowed. If we choose only HTTPS, all HTTP requests will be discarded.

In the *CloudFront distribution with a default CloudFront domain* section of this recipe, we used the default certificate provided by CloudFront (`*.cloudfront.com`) for SSL. This certificate allows us to use HTTPS without the need to create a certificate. In the *CloudFront distribution with a custom domain and ACM certificate* section of this recipe, we specified a wildcard domain name (`*.trainso.io`) for alternate domain names (CNAMEs). This will allow any subdomain to be entered as a CNAME record on the DNS service provider side and forwarded to our web page.

We selected an ACM certificate that was created for a custom domain. We entered a CNAME record on our DNS service provider that points to our CloudFront domain name. The exact steps to add the CNAME record at the DNS service provider server may be specific to a DNS service provider. You can look at the DNS service provider documentation (<https://aws.amazon.com/route53/what-is-dns/>) for more details.

## There's more...

In this recipe, we used our own ACM certificate as a CNAME, which was configured at an outside DNS provider so that it could use a subdomain to access our web page in the S3 bucket. Alternatively, we can use Route 53 to manage the DNS for our domain and point a top-level domain to our CloudFront distribution.

## See also

We can read more about routing traffic to a CloudFront distribution using Route 53 here: <https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/routing-to-cloudfront-distribution.html>.

## Using a WAF

**AWS WAF** (short for **Web Application Firewall**) is a firewall service for monitoring our web traffic. Unlike security groups and **network ACLs (NACLs)**, which only check for ports and IP addresses, AWS WAF can find content matched against predefined signatures that can help detect common attacks, such as SQL injection and cross-site scripting. Currently, we can only use WAF with API Gateway, CloudFront, and ALBs. It cannot be used directly with services such as EC2 or Route 53.

AWS WAF can be used with CloudFront distributions, ALBs, and API Gateway, each having unique characteristics affecting WAF usage. CloudFront is global, meaning WAF rules apply uniformly across all Regions, providing consistent protection for global applications. In contrast, ALBs and API Gateway are regional, so WAF rules must be configured for each region separately, allowing for tailored security policies but increasing management overhead. Using CloudFront with WAF is simpler and can reduce latency due to edge locations, while ALBs and API Gateway offer more granular control but may have regional performance variations and potentially higher costs, due to the need for multiple configurations. Understanding these differences helps optimize security and performance based on your application's architecture and geographic distribution.

## Getting ready

We need the following to successfully complete this recipe:

- A working AWS account, `awssecb-sandbox-1`, and a user, `awssecbadmin1`, as described in the *Technical requirements* section.
- To create a WAF with a CloudFront distribution, we need to create a CloudFront distribution over an S3 bucket by following the *Securing S3 using CloudFront and TLS* recipe of this chapter.

## How to do it...

We can create and configure an AWS WAF for a CloudFront distribution as follows:

1. Log into the AWS Management Console and go to the **WAF & Shield** dashboard.

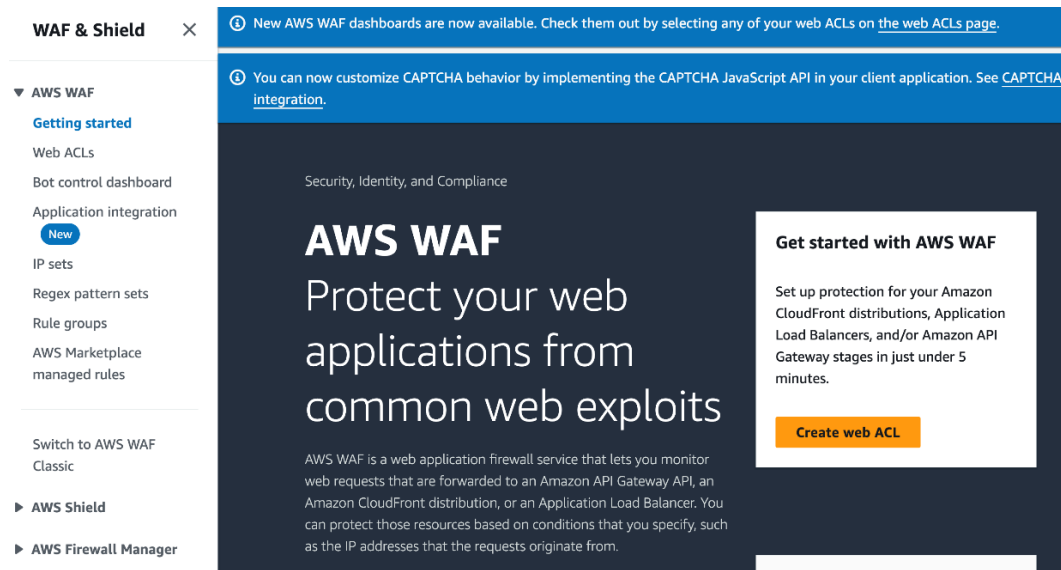


Figure 6.33 – The WAF & Shield dashboard

We should see menu items in the sidebar for **AWS WAF**, **AWS Shield**, and **AWS Firewall Manager**. If AWS decides to separate them, go to the dashboard for AWS WAF and continue with this recipe.

2. Click on **Create web ACL**.

3. In the **Web ACL details** section, for **Resource type**, select **Amazon CloudFront distributions**; for **Name**, enter `cloudericks-webacl`; and optionally, provide a description in the **Description** field. We will use the auto-populated value for the **CloudWatch metric name** field.

### Web ACL details

#### Resource type

Choose the type of resource to associate with this web ACL. Changing this setting will reset the page.

- ☒ **Amazon CloudFront distributions**
- ☐ **Regional resources (Application Load Balancers, Amazon API Gateway REST APIs and AWS AppSync GraphQL APIs)**

#### Region

Choose the AWS Region to create this web ACL in. Changing this setting will reset the page.

Global (CloudFront)

#### Name

cloudericks-webacl

The name must have 1-128 characters. Valid characters: A-Z, a-z, 0-9, - (hyphen), and \_ (underscore).

#### Description - *optional*

Cloudericks Web ACL

The description can have 1-256 characters.

#### CloudWatch metric name

cloudericks-webacl

The name must have 1-128 characters. Valid characters: A-Z, a-z, 0-9, - (hyphen), and \_ (underscore).

Figure 6.34 – Configuring the web ACL

4. Under the **Associated AWS resources** section, click on **Add AWS resources**.
5. On the **Add AWS resources** screen, select the **CloudFront distributions** that we created for this recipe, and click **Add**.
6. Once we are back in the **Associated AWS resources** section, select our CloudFront distribution.
7. For **Web request body inspection**, select **Default**, and then click **Next**.

- 8. On the **Add rules and rule groups** page, expand the **Add rules** dropdown and select **Add my own rules and rule groups**.
- 9. On the **Add my own rules and rule groups** page, select **Rule builder**.

Add my own rules and rule groups [Info](#)

Rule type

☐ IP set  
Use IP sets to identify a specific list of IP addresses.

☒ Rule builder  
Use a custom rule to inspect for patterns including query strings, headers, countries, and rate limit violations.

☐ Rule group  
Use a rule group to combine rules into a single logical set.

Figure 6.35 – Selecting a rule type

- 10. Under **Rule builder**, in the **Rule** section, enter the name `badstring-rule` and set **Type** to **Regular rule**.

Rule

Validate

Name

badstring-rule

The name must have 1-128 characters. Valid characters: A-Z, a-z, 0-9, - (hyphen), and \_ (underscore).

Type

☒ Regular rule

☐ Rate-based rule  
Limits request rates for requests that match your criteria. Applies the action to matching requests when the limit is reached, and removes the action when the rate falls below the limit.

Figure 6.36 – Configuring Rule builder

11. Under the **Statement** section, for **Inspect**, select **Query string**; for **Match type**, select **Contains word**; and for **String to match** enter `badstring`. For **Text transformation**, select **None**.

The screenshot shows the AWS WAF console configuration for a string statement. At the top, there is a header "If a request" followed by a text box containing "matches the statement". Below this is a section titled "Statement". Inside the "Statement" section, there are four configuration options: "Inspect" with a dropdown menu showing "Query string", "Match type" with a dropdown menu showing "Contains word", "String to match" with a text box containing "badstring", and "Text transformation" with a dropdown menu showing "None". Below these options is a button labeled "Add text transformation". At the bottom of the section, there is a note: "You can add up to 10 text transformations."

If a request

**Statement**

Inspect

Match type

String to match

Text transformation  
AWS WAF applies all transformations to the request before evaluating it. If multiple text transformations are added, then text transformations are applied in the order presented below with the top of the list being applied first.

[Add text transformation](#)

You can add up to 10 text transformations.

Figure 6.37 – Setting the string statement

- Under the **Then** section, select **Block** as an action and click **Add rule**.

**Then**

**Action**

Action  
Choose an action to take when a request matches the statements above.

☐ Allow

☒ Block

☐ Count

☐ CAPTCHA

☐ Challenge

► Custom response - optional

► Add label - optional

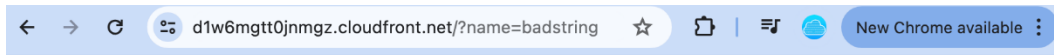
Add labels to requests that match this rule. Rules that are evaluated later in the same web ACL can reference the labels that this rule adds.

Cancel Add rule

Figure 6.38 – Adding a rule

- On the **Add rules and rule groups** page, for **Default web ACL action for requests that don't match any rules**, select **Allow** and click **Next**.
- On the **Set rule priority** page, select our rule and click **Next**.
- On the **Configure metrics** page, select our rule, leave the auto-generated name for the metric, select **Enable sampled requests** for **Request sampling options**, and click **Next**.
- On the **Review and create web ACL** page, review the changes, and click **Create web ACL**. If we go to the CloudFront distribution, it will be deploying. Wait until it is deployed.

17. Run the URL from the browser with a query string that contains `badstring` – for example, `https://d1w6mgtt0jnmgz.cloudfront.net/?name=badstring`. We should get a 403 error, as follows:



## 403 ERROR

### The request could not be satisfied.

Request blocked. We can't connect to the server for this app or website at this time. There might be too much traffic or a configuration error. Try again later, or contact the app or website owner. If you provide content to customers through CloudFront, you can find steps to troubleshoot and help prevent this error by reviewing the CloudFront documentation.

Generated by cloudfront (CloudFront)  
Request ID: nY-RoPx6bVSDDxdFUG7V\_S1CWGAXMQTBVRdQ\_NDQ8NtW13uwEdBnsg==

Figure 6.39 – A 403 error

There will be no error if we do not use the word `badstring` anywhere within our URL.

18. Disassociate the web ACL from our CloudFront distribution, as follows:
  - I. Click on our web ACL.
  - II. Go to the **Associated AWS resources** tab.
  - III. Select our CloudFront distribution.
  - IV. Click on **Remove**.
19. Wait until the status changes to **Deployed**, and run the URL again from a browser with a query string that contains `badstring`. The page should load successfully this time, without any errors.

We can also enable WAF while creating a CloudFront distribution, and we can also disable the protection from the CloudFront distribution.

## How it works...

A web ACL is the primary component within AWS WAF. A web ACL contains one or more rules. Rules contain conditional statements (for example, block access from a range of IP addresses). We added our own rule using the rule builder. The rule builder has an **IF** part and a **THEN** part. The **IF** part contains the condition, while the **THEN** part contains the action that needs to be taken when the condition in the **IF** part is satisfied. In this recipe, we added a simple rule that checks whether the query string contains a string, `badstring`, and blocks such requests.



In the IF part, we can currently inspect the following request components – a header, a single query parameter, all query parameters, the URI path, a query string, a body, and an HTTP method. We can also check whether an IP address is part of an IP set or whether a request originated from a particular country. We created a regular rule in this recipe. We can also create a rate-based rule to set a rate limit for requests from a single user. For example, WAF can block users based on the number of bad requests (4xx errors) they make.

We can create supporting resources such as IP sets, Regex pattern sets, and rule groups, which are used by some of the conditions, from the left sidebar of the WAF dashboard. We can find the AWS Marketplace rule groups using the AWS Marketplace link, which can be found in the left sidebar of the WAF dashboard. Instead of creating our own rules, we can also add an AWS-managed rule group. Currently, there are three categories of managed rule groups in the console – AWS-managed rule groups, Cyber Security Cloud Inc.-managed security groups, and Fortinet-managed rule groups.

## There's more...

At the time of writing, the AWS WAF and AWS Shield services have the same service home page, as we saw in the *How to do it...* section. We saw AWS WAF in detail in this recipe. We will briefly go through some important concepts related to AWS Shield in this section.

AWS Shield is a managed **Distributed Denial of Service (DDoS)** protection service that provides always-on detection and automatic inline mitigations to minimize application downtime and latency, without engaging AWS Support to benefit from DDoS protection.

There are two tiers of AWS Shield – Standard and Advanced. Shield Standard defends against known infrastructure attacks in a network (layer 3) and transport layer (layer 4) that target our website or applications, and it is most effective when used with Amazon CloudFront and Amazon Route 53. AWS Shield Advanced provides higher levels of protection against attacks targeting our applications running on EC2, ELB, CloudFront, AWS Global Accelerator, and Route 53 resources.

## See also

- You can read more about AWS WAF, AWS Shield, and AWS Firewall Manager here: <https://docs.aws.amazon.com/waf/latest/developerguide/what-is-aws-waf.html>.
- You can read more about AWS Shield here: <https://aws.amazon.com/shield/>.

# Monitoring with CloudWatch, CloudTrail, and Config

We've already looked at many aspects of security, such as **Confidentiality, Integrity, and Availability (CIA)** and **Authentication, Authorization, and Accounting (AAA)**. Accounting can be achieved through **continuous monitoring, alerting**, and regular **auditing**. Proper monitoring and alerting can also help in better availability through **auto-remediation**. In this chapter, we will look into **Amazon CloudWatch, AWS CloudTrail, and AWS Config**. CloudWatch is the primary service within AWS for **logging, monitoring**, and alerting. CloudTrail can log AWS API calls. AWS Config can record and evaluate configurations against predefined config rules. We will also learn about **Simple Notification Service (SNS)**, which will allow us to send notifications.

In this chapter, we will cover the following recipes:

- Creating an SNS topic to send emails
- Working with CloudWatch alarms and metrics
- Creating a CloudWatch log group
- Working with EventBridge (previously CloudWatch Events)
- Reading and filtering logs in CloudTrail
- Creating a trail in CloudTrail
- Using Athena to query CloudTrail logs in S3
- Integrating CloudWatch with CloudTrail making use of a CloudFormation template
- Setting up and using AWS Config

## Technical requirements

Before diving into the recipes of this chapter, we need to ensure we have the following requirements and knowledge in place:

- We need an active AWS account to complete the recipes within this chapter. We can use an account that is part of an AWS Organization or a standalone account. I will be using the `awssecb-sandbox-1` account that we created in the *Multi-account management with AWS Organizations* recipe in *Chapter 1*. However, I won't be utilizing any AWS Organizations features, meaning you can follow these steps with a standalone account too.
- For administrative actions, we need a user who has `AdministratorAccess` permission to the AWS account we are working with. This can be an **Identity and Access Management (IAM)** Identity Center user or an IAM user. I will be using the `awssecbadmin1` IAM Identity Center user we created in the *User management and SSO with IAM Identity Center* recipe in *Chapter 1*. However, I won't be utilizing any IAM Identity Center features, meaning you can follow these steps with an IAM user, too, if the user has `AdministratorAccess` permission within the account. You can create an IAM user following the *Setting up IAM, an account aliases, and billing alerts* recipe in *Chapter 1*.

The code files for this chapter are available at <https://github.com/PacktPublishing/AWS-Security-Cookbook-Second-Edition/tree/main/Chapter07>.

## Creating an SNS topic to send emails

In this recipe, we will learn how to create an SNS topic for sending emails. SNS is a managed **publish/subscribe messaging service** and can be used with many endpoints, such as email, **SMS**, **Lambda**, **Simple Queue Service (SQS)**, and more.

### Getting ready

To successfully complete this recipe, we need a working AWS account, and a user as described in the *Technical requirements* section, and a working email address.

### How to do it...

We can configure an SNS topic to send emails as follows:

1. Go to the **Simple Notification Service** service in the console.
2. From the left sidebar, click on **Topics**.
3. Click on **Create topic**.
4. For **Type**, select **Standard**.

5. For the **Name** and **Display name - optional** fields, give meaningful values. I have given the name `MyEmailTopic` and the display name `My Email Topic`.
6. Leave the other options as-is and scroll down to the bottom of the page. Click on **Create topic**.
7. Go to the **Subscriptions** tab of our topic.
8. Click on **Create subscription**.
9. On the **Create subscription** page, set the **Protocol** field to **Email**, and provide an email address that we have access to as the value for the **Endpoint** field.
10. Scroll down and click **Create subscription**. The status of our subscription will be **Pending confirmation** initially.
11. Log in to your email and click on the **Confirm subscription** hyperlink. We should get a **Subscription confirmed** message. If we go back to our subscription in the AWS console and refresh the page, our status should be **Confirmed**.

We have created an SNS topic in this recipe. We will be using this topic in future recipes within this chapter.

## How it works...

In this recipe, we created an SNS topic with an email subscription. To avoid spam, AWS requires us to manually confirm the ownership of the email address by clicking on the confirmation link that was sent to the specified email address.

## There's more...

In this recipe, we selected the **Email** protocol. The following protocols are currently supported: **Email**, **Amazon Kinesis Data Firehose**, **Amazon SQS**, **Amazon Lambda**, **Email-JSON**, **HTTP**, **HTTPS**, **Platform application endpoint**, and **SMS**. **Email-JSON** is different from the **Email** protocol. The **Email-JSON** protocol structures the output as JSON and is useful for services that read emails and process them automatically.

## See also

- You can read more about SNS at <https://www.cloudericks.com/blog/getting-started-with-amazon-sns-service>.
- We can also use **Simple Email Service (SES)** to send email notifications instead of using SNS. SES is a cloud-based email service for sending and receiving emails. It uses the **Simple Mail Transfer Protocol (SMTP)** interface, and all connections to the SMTP endpoint should be encrypted using **Transport Layer Security (TLS)**. The default port for SES is 25, but to avoid **Elastic Compute Cloud (EC2)** throttling email traffic, we can also use port 587 or 2587. Read more about SES at <https://www.cloudericks.com/blog/getting-started-with-amazon-ses>.

# Working with CloudWatch alarms and metrics

In this recipe, we will create a **CloudWatch alarm** using one of the metrics already available. Being the first recipe about CloudWatch, we will also learn about some of the important features of CloudWatch.

## Getting ready

We need the following to successfully complete this recipe:

- A working AWS account and a user as described in the *Technical requirements* section.
- An SNS topic with an email subscription following the *Creating an SNS topic to send emails* recipe from this chapter.

## How to do it...

We can create a CloudWatch alarm using existing metrics as follows:

1. Go to the **CloudWatch** service in the console.
2. Expand **Alarms** from the left sidebar then click on **In alarm**.
3. Click on **Create alarm**.
4. Click on **Select metric**. This should show us all the metrics that are available to us based on the services we use.

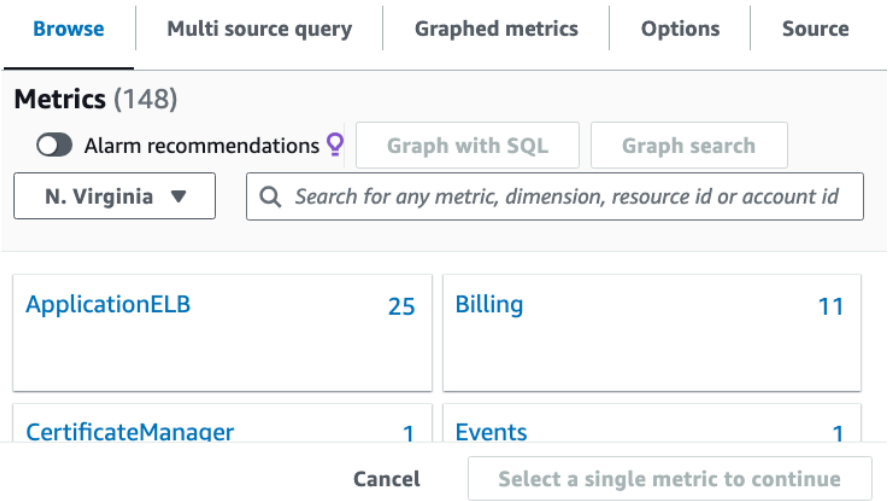


Figure 7.1 – Selecting a metric

5. Click on **Billing**.
6. Click on **By Service**.
7. Select **Amazon EC2** and click on **Select metric**. This will take us to the **Specify metric and conditions** screen.
8. In the **Metric** section of the **Specify metric and conditions** screen, use the defaults shown in the following screenshot:

Graph

This alarm will trigger when the blue line goes above the red line for 1 datapoints within 6 hours.

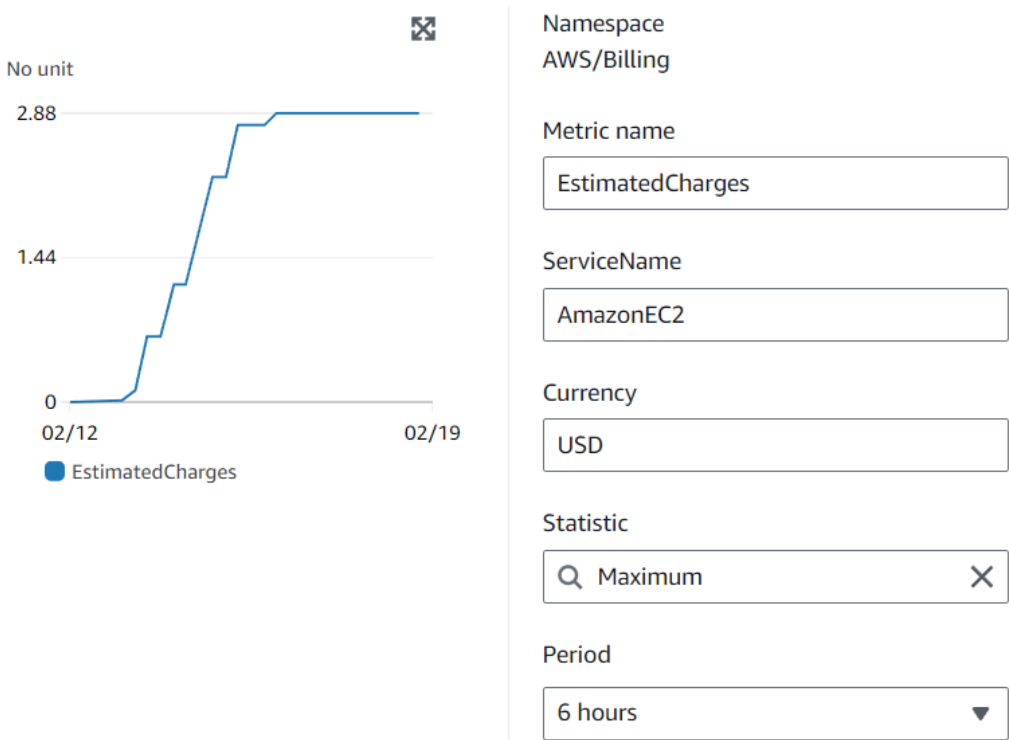


Figure 7.2 – EstimatedCharges metric configuration

9. In the **Conditions** section, define the threshold value as 1 and use the defaults for the other fields.

## Conditions

### Threshold type



**Static**

Use a value as a threshold



**Anomaly detection**

Use a band as a threshold

### Whenever EstimatedCharges is...

Define the alarm condition.



**Greater**

> threshold



**Greater/Equal**

>= threshold



**Lower/Equal**

<= threshold



**Lower**

< threshold

### than...

Define the threshold value.

1

USD

Must be a number

Figure 7.3 – Conditions for the metric

10. Click **Next**.
11. On the **Configure actions** page, select **In alarm** for **Alarm state trigger**. Under **Send a notification to the following SNS topic**, select the **Select an existing SNS topic** option and select the SNS topic we created for this recipe as mentioned in the *Getting ready* section. Leave the other options as-is, as shown in the following screenshot:

## Notification

### Alarm state trigger

Define the alarm state that will trigger this action.

Remove

☒ **In alarm**

The metric or expression is outside of the defined threshold.

☐ **OK**

The metric or expression is within the defined threshold.

☐ **Insufficient data**

The alarm has just started or not enough data is available.

### Send a notification to the following SNS topic

Define the SNS (Simple Notification Service) topic that will receive the notification.

☒ **Select an existing SNS topic**

☐ **Create new topic**

☐ **Use topic ARN to notify other accounts**

### Send a notification to...

🔍 MyEmailTopic



Only topics belonging to this account are listed here.  
All persons and applications subscribed to the selected topic will receive notifications.

### Email (endpoints)

heartin@cloudericks.com - [View in SNS Console](#)

Add notification

Figure 7.4 – Notification configuration for metric

12. Scroll down and click **Next**.
13. Provide **Name** and **Alarm description - optional** values for our alarm. I have set the name to MyEC2BillingAlarm and the alarm description to My EC2 Billing Alarm. Click **Next**.
14. Review the details, scroll down, and click on **Create alarm**. The alarm will appear now on the **Alarms** page. Initially, the **State** field of the alarm will have a value of **Insufficient data**. The state should change to **OK** in some time. Make sure we have a running EC2 instance. When the alarm state changes to the **In Alarm** state, we should get a notification in the email configured with SNS.



**Important note**

I have used a considerably lower value for the threshold and used the EC2 service enough to trigger an alarm. You may use a threshold value as per your requirements.

**How it works...**

We specify the metric while creating a CloudWatch alarm. We can also specify the time period to evaluate the metric, the number of such periods to evaluate before raising an alarm, and the number of data points within the evaluation period, during which the threshold should be crossed, for an alarm to trigger. An alarm's trigger can further trigger additional actions, such as sending a notification through an email, configuring Auto Scaling actions, as well as EC2 actions such as rebooting a failed instance.

A CloudWatch alarm has three states, namely **INSUFFICIENT DATA**, **OK**, and **ALARM**. Until it gets enough data to do the analysis, the state will be **INSUFFICIENT DATA**. After it has enough data points to evaluate and if the result does not cross the threshold over the specified time periods, the state will be **OK**. If the result crosses the threshold in the specified time periods, the alarm is triggered and the CloudWatch alarm goes to the **ALARM** state.

**There's more...**

Let's go through some important concepts related to CloudWatch:

- CloudWatch metrics support alarms, metrics, and dashboards.
- CloudWatch Logs provides log streams and log groups so that we can log data from our applications.
- CloudWatch events support notifications and auto-remediation using AWS Lambda.
- AWS gives us some metrics out of the box and allows us to create custom metrics for our applications.
- Amazon CloudWatch can integrate with AWS X-Ray to provide traces and metrics related to your application's performance.
- CloudWatch integrates with services such as SNS, SQS, AWS Lambda, AWS Auto Scaling, and more for notifications and auto-remediation.
- Under **Alarms** in the left sidebar, we have a dedicated **Billing** option.
- The default EC2 metrics support important operations, including CPU utilization, disk read and write operations, network input and output operations, and status check failures.

## See also

- Read about CloudWatch custom metrics here: <https://www.cloudericks.com/blog/getting-started-with-amazon-cloudwatch-custom-metrics>.
- CloudWatch dashboards provide us with a single place to look into related metrics. Read about creating a dashboard in CloudWatch at <https://www.cloudericks.com/blog/creating-an-amazon-cloudwatch-dashboard>.

## Creating a CloudWatch log group

In this recipe, we will create a **CloudWatch log group** that will be used in other recipes in this book.

### Getting ready

To successfully complete this recipe, we need a working AWS account, and a user as described in the *Technical requirements* section.

### How to do it...

We can create a CloudWatch log group as follows:

1. Go to the **CloudWatch** service in the console.
2. Expand **Logs** from the left sidebar.
3. Click on **Log groups** and click on **Create log group**.
4. Give the log group a name that describes its purpose and leave other settings as default, then click on **Create**.

This will create a new log group for us.

### How it works...

There are not many settings that need to be provided while creating a log group. We can use this log group within other recipes where we log in to CloudWatch. A log group is a group of log streams. A log stream is a sequence of log events from the same source. Log streams within a log group share the same retention, monitoring, and access control settings. We can specify which streams to put into each log group. There is no limit on the number of log streams in a log group.

### There's more...

Log groups may be used from other services and features, such as VPC Flow Logs, as we've seen in the book. Log groups can also be used for our custom-built applications and microservices.

## See also

Read more about CloudWatch logs at <https://www.cloudericks.com/blog/getting-started-with-amazon-cloudwatch-logs>.

## Working with EventBridge (previously CloudWatch Events)

In this recipe, we will learn how to create and use the **EventBridge** service (previously **CloudWatch Events**). CloudWatch events provide us with a near-real-time stream of system events from various AWS resources, and we can create rules to take actions based on the event data.

### Getting ready

We need the following to successfully complete this recipe:

- A working AWS account and a user as described in the *Technical requirements* section.
- An SNS topic with an email subscription following the *Creating an SNS topic to send emails* recipe from this chapter.

### How to do it...

We can work with EventBridge (or CloudWatch events) as follows:

1. Go to the **Amazon EventBridge** service in the console.
2. On the left sidebar, expand **Buses** and click on **Rules**. Click on **Create rule**.

#### Tip

If you are creating events using the CloudWatch service, you have to go to the **Amazon CloudWatch** service from the console. On the left sidebar, select **Rules** under **Events**, and you will be redirected to the EventBridge console to create a rule with a **CloudWatch Events console is now deprecated** message and use the EventBridge console to create and manage event buses and rules. Click on **Create rule**. The remaining steps are the same for creating events.

3. Enter `my-sec-cb-rule-1` as the name and an optional description of `My Sec CB Rule`
  1. Leave the **Event bus** selection as **default**. Lastly, choose **Rule with an event pattern** as the rule type. Click on **Next**.

## Rule detail

**Name**

Maximum of 64 characters consisting of numbers, lower/upper case letters, `.,_-`.

**Description - optional**

**Event bus** [Info](#)

Select the event bus this rule applies to, either the default event bus or a custom or partner event bus.

default

▼

☒ **Enable the rule on the selected event bus**

**Rule type** [Info](#)

☒ **Rule with an event pattern**

A rule that runs when an event matches the defined event pattern. EventBridge sends the event to the specified target.

☐ **Schedule**

A rule that runs on a schedule

Cancel

Next

Figure 7.5 – Rule detail

- Under **Event source**, select **AWS events or EventBridge partner events**.
- Under **Sample event - optional**, leave the default selection for **Sample event type** of **AWS events** as-is, and do not select any sample events.

### Important note

Including sample events is optional but recommended, as they can assist us in writing and testing event patterns or filtering criteria. We can reference a sample event when creating an event pattern or use it to test if the event pattern matches.

- Under **Creation method**, select **Use pattern form**.

7. Under **Event pattern**, do the following, as shown in *Figure 7.6*:

- I. Set **Event source** to **AWS services**.
- II. Set **AWS service** to **EC2**.
- III. Set **Event type** to **EC2 Instance State-change Notification**.
- IV. Set **Event Type Specification 1** to **Any state**.
- V. Set **Event Type Specification 2** to **Any instance**.

**Event pattern** [Info](#)

**Event source**  
AWS service or EventBridge partner as source  
AWS services ▼

**AWS service**  
The name of the AWS service as the event source  
EC2 ▼

**Event type**  
The type of events as the source of the matching pattern  
EC2 Instance State-change Notification ▼

**Event Type Specification 1**  
☒ Any state  
☐ Specific state(s)  
 Specific state(s) ▼

**Event Type Specification 2**  
☒ Any instance  
☐ Specific instance Id(s)

**Event pattern**  
Event pattern, or filter to match the events

```

1 {
2   "source": ["aws.ec2"],
3   "detail-type": ["EC2 Instance State-change Notification"]
4 }

```

Copy Test pattern Edit pattern

Cancel Previous Next

Figure 7.6 – Event pattern section

8. Click on **Next**.
9. Under **Target 1**, do as shown in *Figure 7.7*. For **Target types**, select **AWS service**. Under **Select a target**, select **SNS topic**. For **Topic**, select the topic we created for this recipe, as mentioned in the *Getting ready* section.

**Target 1**

**Target types**  
Select an EventBridge event bus, EventBridge API destination (SaaS partner), or another AWS service as a target.

☐ EventBridge event bus  
☐ EventBridge API destination  
☒ AWS service

Select a target [Info](#)  
Select target(s) to invoke when an event matches your event pattern or when schedule is triggered (limit of 5 targets per rule)

SNS topic ▼

Topic  
MyEmailTopic ▼ ↺

► Additional settings

[Add another target](#) [Cancel](#) [Skip to Review and create](#) [Previous](#) [Next](#)

Figure 7.7 – Target 1 section

10. Click on **Next**.
11. On the **Tags** page, optionally add tags and click **Next**. A tag is a label assigned to an AWS resource, consisting of a key and an optional value. Tags can be used to search and filter resources or track AWS costs.
12. On the **Review and create** page, review all details, and click **Create rule** at the bottom-right corner.

We should now receive a notification email with state change details whenever an EC2 instance changes state. To test this, go to the EC2 dashboard and either create a new instance or change the state of an existing instance.

## How it works...

In this recipe, we selected an event pattern and set the service name to **EC2** and event type to **EC2 Instance State-change Notification** to match EC2 events whose state gets changed. Instead of an event pattern, we can select **Schedule** to invoke our targets on a schedule, just as we can with cron jobs.

We configured to notify any state change. We can also select a specific state, such as **pending**, **running**, **shutting down**, **stopped**, **stopping**, or **terminated**. We also configured to apply this rule to all instances within our account. Instead of doing this, we can select a specific EC2 instance. We selected our SNS topic as the target. When we configure a target, CloudWatch events will provide necessary permissions for targets so that they can be invoked when rules are triggered.

## There's more...

While configuring the event, we selected **SNS topic** as our target. The following is a current list of target types available for us to select: **Amazon Redshift**, **API Gateway**, **AppSync**, **Batch job queue**, **CloudWatch log group**, **CodeBuild project**, **CodePipeline**, **EBS Create Snapshot**, **EC2 ImageBuilder**, **EC2 RebootInstances API call**, **ECS task**, **Firehose stream**, **Glue workflow**, **Incident Manager response plan**, **Inspector assessment template**, **Kinesis stream**, **Lambda function**, **SageMaker pipeline**, **SNS topic**, **SQS queue**, **Step Functions state machine**, **System Manager Automation**, **System Manager OpsItem**, **System Manager Run Command**.

## See also

You can read more about CloudWatch events at <https://www.cloudericks.com/blog/getting-started-with-amazon-cloudwatch-events>.

## Reading and filtering logs in CloudTrail

In this recipe, we will learn how to read and filter **CloudTrail log events** that are automatically generated and made available through the CloudTrail dashboard.

## Getting ready

We need a working AWS account. I will be using the `awsseccb-sandbox-1` account that we created in *Chapter 1*.

## How to do it...

We can check automatically populated event logs in CloudTrail as follows:

1. Log in to the management console and go to the **CloudTrail** service.
2. Click on **Event history** from the left sidebar. This will take us to the **Event history** page.

**Event history (50+)** [Info](#)

Event history shows you the last 90 days of management events.

Lookup attributes

Read-only
Q false

<input type="checkbox"/>	Event name	Event time	User name	Event source	Resource type	Resource name
<input type="checkbox"/>	<a href="#">PutTargets</a>	June 05, 2...	awsseccba...	events.amaz onaws.com	-	-
<input type="checkbox"/>	<a href="#">SetTopicA...</a>	June 05, 2...	awsseccba...	sns.amazona ws.com	AWS::SNS::T...	arn:aws:sns:us-ei
<input type="checkbox"/>	<a href="#">PutRule</a>	June 05, 2...	awsseccba...	events.amaz onaws.com	-	-
<input type="checkbox"/>	<a href="#">ConsoleLo...</a>	June 05, 2...	awsseccba...	signin.amazo naws.com	-	-

Figure 7.8 – Event history

- Under **Lookup attributes**, select **User name** from the dropdown. Type our user's name on the next field (or any name from the list, as seen in *Figure 7.8*) and search for all activity from that user for 10 days using the **Filter by date and time** field. We should now see the filtered list.
- Click on the **Download events** icon at the top-right corner and then click on **Download as CSV** to download the result as a CSV file.

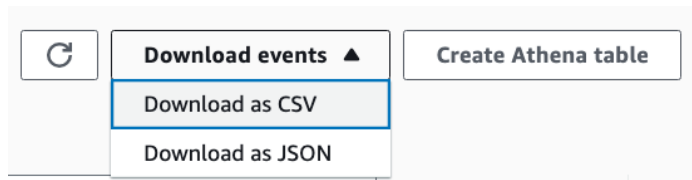


Figure 7.9 – Download events

We can also download the result as a JSON file.



## How it works...

AWS CloudTrail is a service from Amazon that can continuously monitor and log API activities within an AWS account. Without any additional configuration, CloudTrail will log API activity events into our account and event logs are made available for 90 days through the CloudTrail console. After going to the **Events history** page, we can filter based on various criteria and time ranges. We filtered based on the **User name** parameter in this recipe. Apart from **User name**, we can also filter based on the following parameters: **Event name**, **Resource type**, **Resource name**, **Event source**, **Event ID**, **AWS access key**, and **Read only**.

## There's more...

In this recipe, we queried our logs from the console. We can also query logs from the CLI. The following are some important CLI commands for querying CloudTrail logs:

- The `aws cloudtrail lookup-events` command can be used to query the last 90 days of automatically generated event logs. A pagination token is returned if there are more results.
- We can limit the number of items that are returned by the `aws cloudtrail lookup-events` command by specifying the `max-items` option; for example, `aws cloudtrail lookup-events --max-items 10`.
- We can specify a date range using the `start-time` and `end-time` parameters; for example, `aws cloudtrail lookup-events --start-time 2019-01-12 --end-time 2019-10-12`. We can also specify hours, minutes, and seconds with one of these parameters; for example, `--start-time 2019-01-12T00:30:45`.
- We can use the `lookup-attributes` parameter to specify the values of any parameter; for example, `aws cloudtrail lookup-events --lookupattributes "AttributeKey=Username,AttributeValue=i-07d6614e1dec5e537"`.

Let's go through some more important concepts related to CloudTrail logs:

- The CloudTrail service helps us achieve event-driven security by analyzing events and responding to them.
- CloudTrail only records events that involve AWS API calls. Therefore, if an application running on an EC2 instance throws an error, it won't be captured. CloudWatch can be used for logging from applications on EC2 or from Lambda functions.
- By default, a trail will record events in one region. However, we can configure a trail as a multi-region trail. CloudTrail can integrate with other AWS services to provide additional security and compliance. These integrations include CloudWatch for raising alarms, GuardDuty for analyzing patterns, Macie to discover, classify, and protect sensitive data, and so on.

- The current CloudTrail pricing model is as follows: the first tier in each region is free (except **Simple Storage Service (S3)** and Lambda data events). After the free tier, CloudTrail charges us for management events and data events.

## See also

You can read more about CloudTrail at <https://www.cloudericks.com/blog/getting-started-with-aws-cloudtrail>.

## Creating a trail in CloudTrail

In this recipe, we will learn how to create a trail in **CloudTrail** and how to read logs from the associated **S3 bucket**. By default, **CloudTrail API event logs** are made available for 90 days. Data events, such as S3 bucket operations and Lambda invocations, are also not logged by default. To store our logs for more than 90 days, to enable logging data events from S3 or Lambda, and for additional flexibility in searching logs, we can create a trail to log data in an S3 bucket.

## Getting ready

To successfully complete this recipe, we need a working AWS account, and a user as described in the *Technical requirements* section.

## How to do it...

We can create a trail in CloudTrail as follows:

1. Log in to the **CloudTrail** service in the console.
2. Click on **Trails** from the left sidebar.
3. Click on **Create trail**.
4. Provide a **Trial name** value of `aws-sec-cb-events`.
5. Leave **Enable for all accounts in my organization** unchecked. To enable this, we need to log in to the management account of our organization.
6. Under **Storage location**, select **Create a new S3 bucket**.
7. For **Trial log bucket and folder**, enter a unique name for a new S3 bucket and specify a folder (prefix) to store your logs. We may also use the auto-populated value. Remember – bucket names must be globally unique.
8. For **Log file SSE-KMS encryption**, unselect the **Enabled** checkbox.
9. For **Log file Validation**, select **Enabled**.
10. Do not select **Enabled** for **SNS notification delivery**.

11. Do not select **Enabled** for **CloudWatch Logs**.
12. Optionally, add **Tags** and click on **Next**.
13. For **Event type**, select **Management events**.

### Events [Info](#)

Record API activity for individual resources, or for all current and future resources in AWS account. [Additional charges apply](#) 

#### Event type

Choose the type of events that you want to log.

☒ **Management events**

Capture management operations performed on your AWS resources.

☐ **Data events**

Log the resource operations performed on or within a resource.

☐ **Insights events**


Identify unusual activity, errors, or user behavior in your account.

Figure 7.10 – Choosing event type

14. Under **Management events**, select **Read** and **Write** for **API activity** and click on **Next**.

### Management events [Info](#)

Management events show information about management operations performed on resources in your AWS account.

 No additional charges apply to log management events on this trail because this is your first copy of management events.

#### API activity

Choose the activities you want to log.

☒ **Read**

☒ **Write**

☐ **Exclude AWS KMS events**

☐ **Exclude Amazon RDS Data API events**

Figure 7.11 – Configuring management events

15. Click **Next** at the bottom right of the screen.
16. On the **Review and create** page, review all details, and click on **Create trail**. We should see a **Trial successfully created** message.
17. Go to the **Trails** page and click on our trail's S3 bucket name to go to our trail's S3 bucket. We can also manually go to the S3 dashboard and access this bucket. We should see CloudTrail and CloudTrail-Digest folders. Within the folders, we should see sub-folders per region.
18. Go inside the folders until we see the actual log files.

[Amazon S3](#) > [Buckets](#) > [aws-cloudtrail-logs-370598287390-66c52071](#) > [AWSLogs/](#) > [370598287390/](#) > [CloudTrail/](#) > [us-east-1/](#) > [2024/](#) > [06/](#) > [10/](#)

10/ Copy S3 URI

**Objects** | Properties

**Objects (3)** [Info](#)

Refresh Copy S3 URI Copy URL Download Open

Delete Actions ▼ Create folder Upload

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

< 1 > ⚙️

<input type="checkbox"/>	Name ▲	Type ▼	Last modified
<input type="checkbox"/>	<a href="#">370598287390_CloudTrail-us-east-1_20240610T0340Z_SaoUal</a>	gz	June 9, 2024, 21:44:32 (UTC-06:00)

Figure 7.12 – CloudTrail log files

19. Select a file in S3, and from the **Actions** dropdown, click on **Query with S3 Select**.

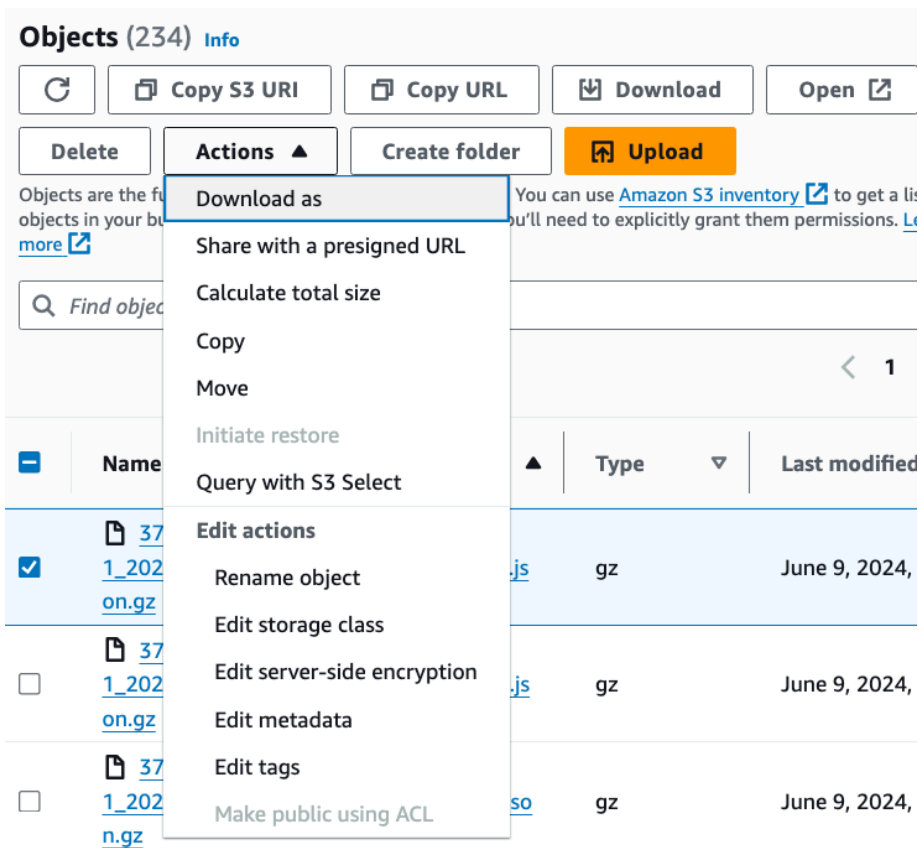


Figure 7.13 – Query with S3 Select

20. Select **JSON** as the format, **Lines** as the JSON content type, and **GZIP** for compression.

## Input settings

### Path

s3://aws-cloudtrail-logs-370598287390-66c52071/AWSLogs/370598287390/CloudTrail/us-east-1/2024/06/10/370598287390\_CloudTrail\_us-east-1\_20240610T0340Z\_5aoUalszjHtODO0U.json.gz

### Size

664.0 B

### Format

- ☐ CSV
- ☒ JSON
- ☐ Apache Parquet

### JSON content type

- ☒ Lines  
Each line in the input data contains a single JSON object
- ☐ Document  
A single JSON object can span multiple lines in the input

### Compression

- ☐ None
- ☒ GZIP
- ☐ BZIP2

Figure 7.14 – Input settings

21. For **Output settings**, select **JSON**.
22. Scroll down, and we should now see the **Structured Query Language (SQL)** query to run. Modify the SQL as required and click on **Run SQL query**.

## SQL query

Add SQL from templates

Run SQL query

Amazon S3 Select supports only the SELECT SQL command. Using the S3 console, you can extract up to 40 MB of records from an object that is up to 128 MB in size. To work with larger files or more records, use the AWS CLI, AWS SDK, or Amazon S3 REST API. For more complex SQL queries, use [Amazon Athena](#).

```
1 /* To create reference point for writing SQL queries, you can display
   the first 5 records of input data by running the following SQL
   query: SELECT * FROM s3object s LIMIT 5 */
2 SELECT * FROM s3object s LIMIT 5
```

SQL Ln 1, Col 1 Errors: 0 Warnings: 0

Figure 7.15 – Run SQL query

We should see the actual result in another textbox, as shown in the following screenshot:

### Query results

Query results are not available after you choose **Close** or navigate away. Choose **Download results** to download a copy of the following query results.

**Download results**

Status

✔ Successfully returned 1 record in 741 ms

Bytes returned: 1169 B

```
1 {
2   "Records": [
3     {
4       "eventVersion": "1.09",
5       "userIdentity": {
6         "type": "AWSService",
7         "invokedBy": "cloudtrail.amazonaws.com"
8       },
```

Figure 7.16 – Query result response partial

23. Click **Download results** to download the result. We can open the downloaded file using any compatible application, such as Microsoft Word.

We created a trail and used it in this recipe. From the **Trails** page, we can go to the configuration page for our trail, and then we can select **Delete** or **Stop logging** if needed.

## How it works...

For storing logs for more than 90 days, we need to create a trail, and trails will send logs into an S3 bucket. In this recipe, we created a multi-region trail. We configured options to log all events. We can also select activities to log. We also saw the option to stop logging for a trail on the trail's configuration page. Stopping logging will stop any new events from being sent to the log, but existing logs will still be available.

We configured management events in AWS CloudTrail to monitor AWS service API activities and track read and write operations, as well as changes to resources and configurations across our AWS account. This setup captures actions such as creating, deleting, modifying, and describing AWS resources, ensuring comprehensive oversight for auditing and compliance.

## There's more...

We did not enable data events and Insights events in this recipe. Data events log operations such as S3 object changes and Lambda function calls, offering detailed resource monitoring, but enabling them increases logging costs. CloudTrail allows logging of data events options, such as **S3**, **DynamoDB**, **Lambda**, **Amazon Q apps**, **Amazon Q Business application**, **Amazon Q Business data source**, **Amazon Q Business index**, **Amazon Verified Permissions**, **AWS AppConfig**, **AWS Cloud Map namespace**, **AWS Cloud Map service**, **B2B Data Interchange**, **Bedrock agent alias**, **Bedrock knowledge base**, **Cassandra table**, **CloudFront KeyValueStore**, **CloudTrail channel**, **CloudWatch metric**, **CodeWhisperer**, **CodeWhisperer customization**, **Cognito Identity Pools**, **DynamoDB Streams**, **EBS direct APIs**, **EMR write-ahead log workspace**, **FinSpace**, **Guard Duty detector**, **IoT certificate**, **IoT Greengrass component version**, **IoT Greengrass deployment**, **IoT SiteWise asset**, **IoT SiteWise time series**, **IoT thing**, **IoT TwinMaker entity**, **IoT TwinMaker workspace**, **Kendra Ranking**, **Kinesis stream**, **Kinesis stream consumer**, **Kinesis video stream**, **Lake Formation**, **Machine Learning MIModel**, **Managed Blockchain**, **Managed Blockchain network**, **Managed Blockchain Query**, **MedicalImaging data store**, **Neptune Graph**, **Private CA Connector for Active Directory**, **Private CA Connector for SCEP**, **RDS Data API - DB Cluster**, **S3 Access Point**, **S3 Object Lambda**, **S3 Outposts**, **SageMaker endpoint**, **SageMaker Feature Store**, **SageMaker metrics experiment trial component**, **SNS platform endpoint**, **SNS topic**, **SQS**, **Step Functions state machine**, **Supply Chain**, **SWF domain**, **Systems Manager**, **Systems Manager managed node**, **Thin Client Device**, **Thin Client Environment**, **Timestream database**, **Timestream table**, and **X-Ray trace**. These options provide additional insight into activities within our AWS resources.

Insight events logging focuses on detecting abnormal usage patterns or potential security threats related to API calls that modify or manage AWS resources. Insight events cannot be selected alone but should be paired with management events to ensure comprehensive logging coverage and contextual analysis of AWS API activity for security, compliance, and operational purposes.

In this recipe, we queried logs in S3 directly from the S3 console. For more flexibility when querying CloudTrail logs in S3, we can use Amazon Athena. We will learn how to use Amazon Athena to query CloudTrail logs in the *Using Athena to query CloudTrail logs in S3* recipe of this chapter.

## See also

Explore more about CloudTrail at <https://www.cloudericks.com/blog/deep-dive-into-aws-cloudtrail>.

## Using Athena to query CloudTrail logs in S3

In this recipe, we will learn how to use **Amazon Athena** to query CloudTrail logs. Using Athena to query CloudTrail logs provides us with greater flexibility. For example, we cannot filter based on an account ID from the CloudTrail console when multiple accounts are sending logs to CloudTrail's S3 bucket. However, we can use Athena to query for logs from CloudTrail's S3 bucket based on the account ID.




## Getting ready

We need the following to successfully complete this recipe:

- A working AWS account and a user as described in the *Technical requirements* section.
- A trail in CloudTrail following the *Creating a trail in CloudTrail* recipe from this chapter.
- If we are new to Athena, before we can run our queries, we should set up a query result location in Amazon S3, as follows:
  - I. Create a bucket for query results. I've named my bucket `aws-sec-cb2-query-results`. Choose a unique name for your bucket.
  - II. Go to the **Athena** service in the management console.
  - III. Go to **Query editor tabs**. If we are new to Athena, we should see a warning stating **Before you run your first query, you need to set up a query result location in Amazon S3**. Click on **Edit settings**.
  - IV. Browse S3 and choose the bucket we created for query results.
  - V. Leave other options as-is and click on **Save**.

### Location of query result - optional

Enter an S3 prefix in the current region where the query result will be saved as an object.



**You can create and manage lifecycle rules for this bucket**

Use Amazon S3 lifecycle rules to store your query results and metadata cost effectively or to delete them after a period of time. [Learn more](#)

[Lifecycle configuration](#)

**Expected bucket owner - optional**

Specify the AWS account ID that you expect to be the owner of your query results output location bucket.

Enter AWS account ID

☐ **Assign bucket owner full control over query results**  
 Enabling this option grants the owner of the S3 query results bucket full control over the query results. This means that if your query result location is owned by another account, you grant full control over your query results to the other account.

☐ **Encrypt query results**


Figure 7.17 – Setting query results location

## How to do it...


We can set up Athena and query CloudTrail logs as follows:

1. Log in to the **CloudTrail** service in the management console.
2. Click on **Event history** from the left sidebar. This will take us to the **Event history** page.
3. Click on **Create Athena table**, as we saw in *Figure 7.9*.
4. For **Storage location**, select the S3 bucket of our trail.

**Create a table in Amazon Athena** ✕

You can use Amazon Athena to analyze events that are stored in a trail's Amazon S3 bucket. Athena is an interactive query service that helps you analyze data in S3 buckets by using standard SQL. Athena charges for running queries.[Learn more](#) 

Storage location

aws-cloudtrail-logs-370598287390-66c52071 

Choose an S3 bucket that contains CloudTrail log files

Athena table name

cloudtrail\_logs\_aws\_cloudtrail\_logs\_370598287390\_66c52071

This name is auto-generated. You can rename it in Amazon Athena.

**Athena table query** Copy

```
1 CREATE EXTERNAL TABLE
  cloudtrail_logs_aws_cloudtrail_logs_370598287390_66c52071
2   eventVersion STRING,
3   userIdentity STRUCT<
```

Figure 7.18 – Creating Athena table

5. Scroll down and click on **Create table**. We should see a success message that the table has been created.
6. Now, go to **Athena** service in the management console.
7. Go to the **Query editor** tab. We should see our table under **Tables**, on the left, and a query editor window on the right. It might take some time for the table to be created and appear on the Athena dashboard after we've initiated its creation from the CloudTrail dashboard. We can manually refresh the table list using the refresh icon in the left sidebar.

8. Click on the button with three dots next to our table and click on **Preview Table**. A sample query will be created that we can modify so that it meets our needs.

I have set the limit to 2 in the query as follows: `SELECT * FROM "default"."cloudtrail_logs_aws_cloudtrail_logs_370598287390_66c52071" limit 2;`

9. Click **Run**.
10. Click on the **Download results** icon on the top right of the page to download the results in CSV format.

If we go to the S3 bucket that we created for storing results, which is `aws-sec-cb2-query-results` in my case, we should see the query results saved there too.

## How it works...

In this recipe, we used Amazon Athena to query CloudTrail logs in S3. Athena uses queries based on SQL and creates virtual tables. If we are new to Athena, before we can run our queries, we should set up a query result location in Amazon S3. We created an Athena table from the CloudTrail dashboard. Then, we went to Athena and ran a preview query. We modified the query and executed it. Finally, we exported the results into a CSV file from the results screen.

## There's more...

Let's quickly go through some important concepts related to Amazon Athena:

- Athena is a query service provided by AWS to analyze data in Amazon S3 using SQL.
- Athena can only query S3 data and not CloudTrail directly.
- Amazon Athena now supports federated queries. We can run SQL queries across data stored in relational, non-relational, object, and custom data sources and then store the result in Amazon S3. At the time of writing, this feature is in preview. Athena is serverless. We do not have to set up any infrastructure and we only pay for the queries we run.
- Athena, integrated with AWS Glue, allows us to crawl data sources, populate table and partition definitions, and even maintain schema versioning.

## *Cross-account CloudTrail logging*

With cross-account CloudTrail logging, we can store logs in a separate account from the account in which logs are generated. By storing logs in a separate account, we isolate the logs from the source account, which will prevent anyone with access to the source account from tampering with the logs. We can provide account-level access to the log account to a limited set of people. Sending logs from multiple accounts to a single account also provides a central place for us to query logs.

To set up cross-account CloudTrail logging, we need two AWS accounts: a log account (where logs are stored) and a logger account (which sends the logs). If we are using AWS Organizations, we can enable CloudTrail across accounts from the management account by selecting **Enable for all accounts in my organization** while creating a trail. An organization's trail is then created in all member accounts, which means we don't have to modify the bucket policy. However, enabling this option may lead to additional charges if the member accounts already have trails since only the first trail within a region is free.

In this recipe, we did not select the option to enable CloudTrail across accounts using AWS Organizations. To set up cross-account CloudTrail logging without using AWS Organizations, first configure a CloudTrail trail in the log account and set up an S3 bucket to store the logs. Modify the bucket policy to allow the CloudTrail service and specific IAM roles from both accounts to write logs to the bucket. Then, create a trail in the logger account, specifying the log account's S3 bucket as the storage location. Verify that logs from the logger account are being correctly stored in the specified S3 bucket within the log account. This setup isolates the logs from the source account and centralizes log storage, enhancing security and manageability.

## See also

- Read more about Amazon Athena at <https://www.cloudericks.com/blog/getting-started-with-amazon-athena>.
- Read more about cross-account CloudTrail logging at <https://www.cloudericks.com/blog/getting-started-with-cross-account-cloudtrail-logging>.

## Integrating CloudWatch with CloudTrail making use of a CloudFormation template

In this recipe, we will learn how to integrate CloudWatch with CloudTrail. Once integrated, we can create **metric filters** and alarms within CloudWatch based on CloudTrail logs. We will also learn how to use the CloudFormation template provided by AWS to create a set of alarms within CloudWatch that uses CloudTrail logs.

## Getting ready

We need the following to successfully complete this recipe:

- A working AWS account and a user as described in the *Technical requirements* section.
- A trail, by following the *Creating a trail in CloudTrail* recipe of this chapter.

### Important note

You can also perform the steps in this recipe while creating a trail by making minor modifications to them.

## How to do it...

We can integrate CloudWatch with an existing trail as follows:

1. Go to the **CloudTrail** service in the management console.
2. Click on **Trails** from the left sidebar.
3. Click on the name of our trail to go to the trail's configuration page.
4. Scroll down to the **CloudWatch Logs** section and click on **Edit**.
5. For **CloudWatch Logs**, select the **Enabled** checkbox.
6. For **Log group**, select **New** and leave the log group name as-is, which is `aws-cloudtrail-logs-370598287390-b9277b0a` in my case. We can also use an existing log group using the **Existing** option.
7. For **IAM Role**, select **New**, provide a role name, and click on **Save changes**.
8. Download the CloudFormation template from [https://s3-us-west-2.amazonaws.com/awscloudtrail/cloudwatch-alarms-for-cloudtrail-api-activity/CloudWatch\\_Alarms\\_for\\_CloudTrail\\_API\\_Activity.json](https://s3-us-west-2.amazonaws.com/awscloudtrail/cloudwatch-alarms-for-cloudtrail-api-activity/CloudWatch_Alarms_for_CloudTrail_API_Activity.json) and save it locally.
9. Go to the **CloudFormation** service in the management console and click on **Create stack**.
10. For **Prepare template**, select the **Choose an existing template** option.

### Prerequisite – Prepare template

#### Prepare template

Every stack is based on a template. A template is a JSON or YAML file that contains configuration information about the AWS resources you want to include in the stack.

<input checked="" type="radio"/> <b>Choose an existing template</b> Upload or choose an existing template.	<input type="radio"/> <b>Use a sample template</b> Choose from our sample template library.	<input type="radio"/> <b>Build from Application Composer</b> Create a template using a visual builder.
---	--	---

Figure 7.19 – Prepare template

11. Under **Specify template**, choose the **Upload a template file** option, and using the **Choose file** option, upload the template we downloaded in *Step 8* of this section.

## Specify template [Info](#)

A template is a JSON or YAML file that describes your stack's resources and properties.

### Template source

Selecting a template generates an Amazon S3 URL where it will be stored.

☐ Amazon S3 URL

Provide an Amazon S3 URL to your template.

☒ Upload a template file

Upload your template directly to the console.

☐ Sync from Git - *new*

Sync a template from your Git repository.

### Upload a template file

 Choose file

CloudWatch\_Alarms\_for\_CloudTrail\_API\_Activity.json



JSON or YAML formatted file

S3 URL: [https://s3.us-east-1.amazonaws.com/cf-templates-1w3mwky7gce2-us-east-1/2024-06-17T193713.298Zh62-CloudWatch\\_Alarms\\_for\\_CloudTrail\\_API\\_Activity.json](https://s3.us-east-1.amazonaws.com/cf-templates-1w3mwky7gce2-us-east-1/2024-06-17T193713.298Zh62-CloudWatch_Alarms_for_CloudTrail_API_Activity.json)

[View in Application Composer](#)

Figure 7.20 – Specify template

12. Scroll down and click **Next**.
13. On the **Specify stack details** page, provide a stack name and email address so that we'll be notified when an API activity has triggered an alarm.
14. Leave the cloud trail log group name as-is and click on **Next**.
15. On the **Configure stack options** page, leave the defaults as-is and click on **Next**.
16. On the **Review and create** page, review everything and click on **Submit**. Wait for our CloudFormation stack to be created.
17. After the CloudFormation stack has been successfully created, we will receive an email asking us to verify our email address. To start receiving email notifications when alarms are activated, click on the **Confirm subscription** link provided in the email.
18. If we go to the **Alarms** page in **CloudWatch**, we will be able to view the new alarms that were created. We can wait until the alarm state changes to **OK** or **ALARM** and play around with these alarms to learn more about them.

## How it works...

In this recipe, we integrated CloudWatch with CloudTrail through our trail's settings. CloudTrail asked us for permission to deliver CloudTrail events associated with API activity in our account to our log group. We allowed this from the console. The following permissions were granted:

- `CreateLogStream`, to create a log stream in the log group we specified
- `PutLogEvents`, to deliver CloudTrail events to the log stream

We used the CloudFormation template provided by AWS to set up some CloudWatch alarms for security and network-related API activity. If we delete the CloudFormation stack, all the alarms will also be deleted.

AWS uses SNS to send notifications and created an SNS topic subscription for us. We had to confirm the email subscription by verifying our email address since SNS will not send notifications until we manually confirm the subscription.

## There's more...

In this recipe, we integrated CloudWatch with CloudTrail and used the CloudFormation template provided by Amazon to create some alarms that are related to security and network-related API activity. You can add these alarms to a dashboard following the reference in the *See also* section of the *Working with CloudWatch alarms and metrics* recipe from this chapter.

## See also

Read more about CloudFormation at <https://www.cloudericks.com/blog/aws-cloudformation-for-absolute-beginners>.

## Setting up and using AWS Config

In this recipe, we will learn how to set up and use AWS Config. We can use Config to record and evaluate configurations of our AWS resources. We can create rules that define security standards and find out about resources that do not comply with security standards. Config also supports the auto-remediation of problems whenever they are detected.

## Getting ready

We need the following to successfully complete this recipe:

- A working AWS account and a user as described in the *Technical requirements* section.

- If you want to add an SNS topic for notifications, you can create an SNS topic with an email subscription by following the *Creating an SNS topic to send emails* recipe of this chapter.
- For testing, we need at least one IAM user without **multi-factor authentication** (MFA) enabled.

## How to do it...

First, we will set up AWS Config for the first time, and then we will see how to use AWS Config. Let us get started:

1. When we log in to the **AWS Config** service in the management console for the first time, we will see a **Getting started** page. Click on **Get started**, and we will be taken to the **Settings** page.
2. In the **Recording method** section, for **Recording strategy**, select **All resource types with customizable overrides**.

### Recording method

#### Recording strategy

Customize AWS Config to record configuration changes for all supported resource types, or for only the supported resource types that are relevant to you. Globally recorded resources (RDS global clusters and IAM users, groups, roles, and customer managed policies) may be recorded in more than this Region. [Learn more](#)  You are charged based on the number of configuration items recorded. [Pricing details](#) 

☒ **All resource types with customizable overrides**

AWS Config will record all current and future supported resource types in this Region. You can override the recording frequency for specific resource types or exclude specific resource types from recording.

☐ **Specific resource types**

AWS Config will only record the resource types that you specify.

Figure 7.21 – AWS Config recording method

3. For **Recording frequency**, select **Continuous recording**.

### Default settings

#### Recording frequency

Configure the default recording frequency for all current and future supported resource types. It impacts the cost to your bill. [Pricing details](#) 

☒ **Continuous recording**

Record configuration changes continuously whenever a change occurs.

☐ **Daily recording**

Receive configuration data once every day only if a change has occurred.

Figure 7.22 – Recording frequency





4. Under the **Override settings** section, click on **Remove**.

**Override settings**

Resource types to override [Info](#)

Override the recording frequency for specific resource types, or exclude specific resource types from recording. If you change the recording frequency for a resource type or stop recording a resource type, the configuration items that were already recorded will remain unchanged.

 "All globally recorded IAM resource types" are initially excluded from recording to help you reduce costs. Choose **Remove** to remove the override and include these resources in your recording. 

Resource type	Override
All globally recorded IAM resou... ▼	Exclude from recording ▼

Remove

Figure 7.23 – Override settings

5. In the **Data governance** section, select **Create AWS Config service-linked role**. Alternatively, we may choose a role from our account.
6. In the **Delivery method** section, select **Create a bucket** and give an S3 bucket name.

**Delivery method**

Amazon S3 bucket

☒ Create a bucket

☐ Choose a bucket from your account

☐ Choose a bucket from another account

Ensure appropriate permissions are available in this S3 bucket's policy. [Learn more](#)

S3 Bucket name (required)

config-bucket-370598287390

Prefix (optional)

/AWSLogs/370598287390/Config/us-east-1

Amazon SNS topic

☐ Stream configuration changes and notifications to an Amazon SNS topic.  
If you choose email as the notification endpoint for your SNS topic, this can cause a high volume of email. [Learn more](#)

Figure 7.24 – Delivery method section

7. In the **Amazon SNS topic** section, select **Stream configuration changes and notifications to an Amazon SNS topic**, then select **Choose a topic from your account**, and finally select the topic we created as mentioned in the *Getting ready* section. Alternatively, we may create a topic or choose a topic from another account.

#### Amazon SNS topic

- ☒ **Stream configuration changes and notifications to an Amazon SNS topic.**

If you choose email as the notification endpoint for your SNS topic, this can cause a high volume of email. [Learn more](#) .

☐ Create a topic

☒ Choose a topic from your account

☐ Choose a topic from another account

Ensure appropriate permissions are available in this SNS topic's policy. [Learn more](#) .

#### SNS topic name

Mytopic ▼

Figure 7.25 – Choosing an Amazon SNS topic

8. At the bottom of the page, click **Next**.
9. On the **AWS Managed Rules** page, search for and select the `iam-user-mfa-enabled` rule. Click **Next**. We can add more rules if we want. We can also add rules after completing the setup process.
10. On the **Review** page, review the changes and click **Confirm** at the bottom-right corner. We will be redirected to the AWS Config dashboard. In the dashboard, we can see **Conformance Packs by Compliance Score**, **Compliance status**, **Noncompliant rules by noncompliant resource count**, **Resource inventory**, **AWS Config usage metrics**, and **AWS Config success metrics**.
11. If we have an IAM user without MFA enabled, as discussed in the *Getting ready* section, we should see that user as not compliant with our `iam-user-mfa-enabled` rule.

Please note that it will take some time for the non-compliant resources to be shown in the dashboard. We can also do the following actions for the rule: **Manage remediation**, **Re-evaluate**, **Delete results**, and **Delete rule**.

## How it works...

In this recipe, we set up AWS Config on our account. We selected **Record all resources supported in this region** and **Include global resources** (for example, AWS IAM resources) against **All resources** in order to record all resources across all regions. We can also configure recording for specific resources by unchecking **Record all resources supported in this region** and selecting the resources we want in the **Specific types** field.

We enabled SNS notifications to receive email notifications by selecting an SNS topic with an email subscription from our account. We can also choose an SNS topic from another account by selecting the **Choose a topic from another account** option. In the **AWS Config role** section, we selected the **Create AWS Config service-linked** role. This role grants read-only access for Config to our AWS resources so that we can record configuration information. The role also grants permission to send information to S3 and SNS.

Here, we selected the `iam-user-mfa-enabled` rule, which is a periodic rule. Periodic rules are run periodically, whereas non-periodic rules (configuration changes-based rules) are run immediately when a change is made to an associated configuration.

## There's more...

Let's go through some important concepts related to AWS Config:

- Some of the checks that we can do with AWS Config include the following: check if MFA is enabled, check if S3 buckets are not public, databases are encrypted, VPC flow logs are enabled, and so on.
- We can write our own custom rules using AWS Lambda.
- AWS Config can perform auto-remediation actions for a rule. For example, we can change the configuration of an EC2 instance based on a rule. However, AWS may stop and restart the EC2 instance, so we need to consider possible downtime.
- To configure auto-remediation from the new console, we can go to our rule, click on the **Actions** dropdown, and select **Manage remediation**.
- We can use the same set of Config rules across multiple accounts to ensure they all follow a common set of rules.
- We can create an aggregator in AWS Config to view the AWS resource inventory or Config rule compliance status for multiple accounts across all regions. In the current console, we can create an aggregator by going to the **Aggregate** view from the left sidebar and then clicking on **Add aggregator**.
- AWS Config is charged based on the number of rules evaluations that are recorded.

---

The steps for creating a custom rule can be summarized as follows:

1. Create an IAM role that can be used by a Lambda with the required permissions. To report back to AWS Config, we need to provide `AWSConfigRulesExecutionRole`. To send logs to CloudWatch, we will need to add `AWSLambdaBasicExecutionRole`. Finally, we need to give access to the service it will monitor (for example, `AmazonS3ReadOnlyAccess` for accessing S3).
2. Create a Lambda by selecting the IAM role we created in the previous step using any of the supported programming languages.
3. Write some code within Lambda in order to evaluate the service parameters that we are monitoring (for example, S3 bucket properties), update a `ResultToken` object per evaluation, and return a list of `ResultToken` objects back to Config. The `ResultToken` objects should consist of the following information: `ComplianceResourceType` (for example, `AWS::S3::Bucket`), `ComplianceResourceId` (for example, bucket name), `ComplianceType` (`COMPLIANT` or `NON_COMPLIANT`), and `OrderingTimestamp`.
4. Within the Config dashboard, we can go to **Rules**, then to **Add rule**, and select **Add Custom rule**. The exact screen names of these options may be different when you do this.
5. Set the **Trigger type** field to **Configuration changes** or **Periodic**.
6. Next, we can choose a remediation action or notification for our rule.
7. Click **Save**. Our rule should be listed along with the other rules on the **Rules** page.

## See also

Read more about AWS Config at <https://www.cloudericks.com/blog/getting-started-with-aws-config.vp>.



# Compliance with GuardDuty, Macie, Inspector, and Analyzer

Regularly checking for compliance within our account and being notified if anything is not compliant is an important step toward keeping our account secure. In this chapter, we will learn about some services within AWS that can help us in checking compliance, with the help of additional intelligence and rules. We will learn about **Amazon GuardDuty**, **Amazon Macie**, and **Amazon Inspector**, which use machine learning and advanced algorithms to help us check for compliance. AWS Config is another service that can help with compliance, but we already covered it in *Chapter 7*.

This chapter includes the following recipes:

- Setting up and using Amazon GuardDuty
- Aggregating findings from multiple accounts in GuardDuty
- Setting up and using Amazon Macie
- Setting up and using Amazon Inspector
- Setting up and using AWS Security Hub
- Using IAM Analyzer to inspect unused access

## Technical requirements

Before diving into the recipes of this chapter, we need to ensure that we have the following requirements and knowledge in place:

- We need an active AWS Account to complete the recipes within this chapter. If we are using an AWS Organization, we can use the management account of the Organization as we will be configuring many things at the AWS Organization level in this chapter. I will be using the `aws-sec-cookbook-1` account that we created in the *Multi-account management with AWS Organizations* recipe in *Chapter 1*.

- For administrative actions, we need a user who has **AdministratorAccess** permission to the AWS account we are working with.

Code files for this chapter are available at <https://github.com/PacktPublishing/AWS-Security-Cookbook-Second-Edition/tree/main/Chapter08>.

## Setting up and using Amazon GuardDuty

In this recipe, we will learn how to set up and use Amazon GuardDuty. GuardDuty analyzes data from sources such as CloudTrail management logs, VPC flow logs, and Route 53 DNS logs, and uses machine learning, anomaly detection, and integrated threat intelligence to find malicious activities and unauthorized behavior. Note that if you use another DNS resolver, such as OpenDNS or GoogleDNS, or if you set up your own DNS resolvers, GuardDuty cannot access and process data from these sources. GuardDuty can be integrated with CloudWatch and SNS to raise alarms and send notifications. GuardDuty can also aggregate data from multiple accounts.

### Getting ready

To successfully complete this recipe, we will need a working AWS account and a user, as described in the *Technical requirements* section. We will also need an S3 bucket.

### How to do it...

If we are using GuardDuty for the first time, we need to first enable GuardDuty. Let us get started:

1. Go to the **GuardDuty** service in the AWS management console.
2. If we are using GuardDuty for the first time, we should see options such as the following. If GuardDuty is already enabled, proceed to *Step 4*.

#### Try threat detection with GuardDuty

- ☒ **Amazon GuardDuty - all features**  
Experience threat detection capabilities in your AWS environment.
- ☐ **GuardDuty Malware Protection for S3 only**  
Detect malicious files that are newly uploaded to your Amazon S3 buckets. You don't need to enable Amazon GuardDuty.

**Get started**

Figure 8.1 – Enabling GuardDuty

3. Select **Amazon GuardDuty - all features** and click on **Get started**. We should now see the **Welcome to GuardDuty** screen; click on **Enable GuardDuty**.
4. Click on **Findings** in the left sidebar on the **GuardDuty** dashboard. We can see the findings categorized as **High**, **Medium**, and **Low** here. As we have just enabled it, you may not see any findings initially.

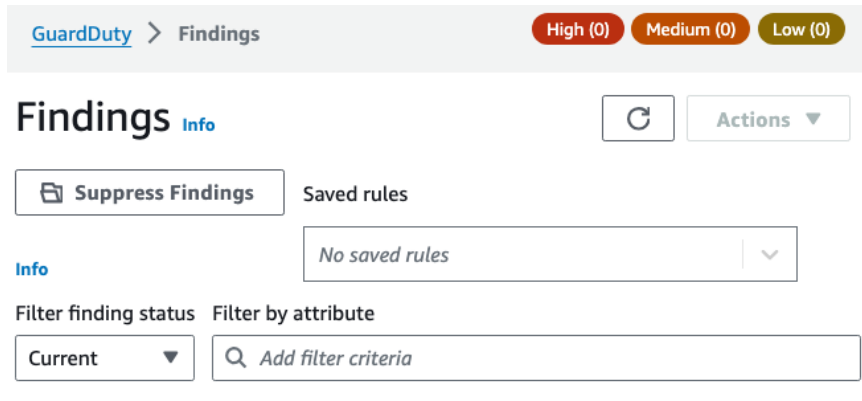


Figure 8.2 – The Findings screen

5. We can click on any of the events to see additional information provided about that finding. We can then scroll down to the **Action**, **Target**, and **Additional information** sections to learn more about the event.

Next, we will whitelist and blacklist IPs in GuardDuty.

### ***Whitelisting and blacklisting IPs in GuardDuty***

We can whitelist and blacklist IPs in GuardDuty as follows:

1. Create and upload the following two files, one with trusted IPs and one with threat lists, and upload them into an S3 bucket:
  - `trusted-ips.txt`: This is a text file with the list of IPs and CIDR ranges we want to trust. Each IP or CIDR range should be on its own line.

```
trusted-ips - Notepad
File Edit Format View Help
192.0.2.0/24
198.51.100.0/24
203.0.113.0/24
|
```

Figure 8.3 – The trusted-ips file



- `threat-lists.txt`: This is a text file with the list of IPs and CIDR ranges we want to add to the suspicious IP list. Each IP or CIDR range should be on its own line.

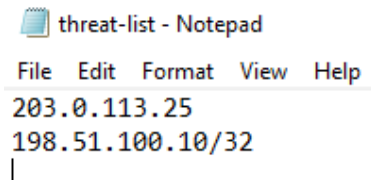


Figure 8.4 – The threat-list file

2. Now, go back to the **GuardDuty** dashboard. Click on **Lists** in the left sidebar.
3. Under **Trusted IP lists**, click on **Add a trusted IP list**.
4. In the pop-up screen named **Add a trusted IP list**, give a name to our list in the **List name** field. For **Location**, enter the S3 URL of your trusted IP list, which is of the `https://myguarddutydemo.s3.amazonaws.com/trusted-ips.txt` form. For **Format**, select **Plaintext**. Then select the **I agree** checkbox. Finally, click on **Add list**. We should now see our list under **Trusted IP lists**.
5. Click the checkbox beside our list. Click on the **Action** dropdown and click on **Activate**. We should see a success message indicating that the list has been activated. It may take some time for the changes to take effect.
6. Under **Threat IP lists**, click on **Add a threat IP list**. We should see a pop-up screen named **Add a threat IP list**.
7. Repeat *Steps 3 to 5* but add our threat list instead of a trusted IP list. We should see our list under **Threat IP lists**.
8. To verify that the trusted and threat IP lists are working correctly, we can simulate traffic from IPs in our lists and check the GuardDuty findings:
  - **Whitelisted IPs (Trusted IPs)**: Generate traffic from an IP address that is in our `trusted-ips.txt` file. Monitor GuardDuty findings to ensure that this traffic is not flagged as suspicious or malicious.
  - **Blacklisted IPs (Threat IPs)**: Generate traffic from an IP address that is in our `threat-lists.txt` file. Check the GuardDuty findings to confirm that this traffic is flagged as suspicious or malicious.
9. Review the GuardDuty dashboard and findings to ensure that the whitelisted IPs are not generating any alerts and that the blacklisted IPs are flagged appropriately. This will confirm that our lists are being used effectively by GuardDuty.

In this section, we whitelisted and blacklisted a few IP addresses by adding them to the trusted IP lists and threat lists, respectively. We will learn more about how they work in the *How it works...* section of this recipe.

## How it works...

We first enabled GuardDuty. When we enabled GuardDuty, we granted it permissions to analyze various logs and data sources, such as VPC Flow logs, AWS CloudTrail management event logs, DNS query logs, AWS CloudTrail S3 data event logs, EKS audit logs, Lambda network activity logs, and RDS login activity logs, to generate security findings, as you should have seen while doing *Step 3* of the *How to do it...* section for this recipe. Additionally, GuardDuty can analyze **Elastic Block Storage (EBS)** volume data to detect malware. When we enable GuardDuty in a supported region for the first time, our account is automatically enrolled in a 30-day free trial, which may also include some protection plans by default.

By default, when we first enable GuardDuty, all protection plans are activated, except for Runtime Monitoring and Malware Protection for S3. These can be enabled through the GuardDuty console or APIs. Usage of GuardDuty's Malware Protection and Runtime Monitoring services is governed by the Amazon GuardDuty Service Terms. We can suspend or disable GuardDuty or any specific protection plan at any time to stop it from processing and analyzing data, events, and logs. However, suspending or disabling GuardDuty does not affect Malware Protection for S3. To stop malware scanning of our S3 buckets, we must individually delete the Malware Protection plan for each bucket. Note that GuardDuty does not manage or provide access to the logs and data it analyzes; we can configure these data sources through their respective consoles or APIs.

For testing purposes, we generated sample findings from the GuardDuty console; GuardDuty generated 54 sample events (in my case – it could vary for you since GuardDuty adds and deprecates finding types throughout the year). GuardDuty events are categorized into three severity levels, from lowest to highest, denoted by blue, yellow, and red icons, where blue is the least severe and red is the most severe. Once we click on a finding, we will get additional information about the finding.

We whitelisted and blacklisted a few IP addresses by adding them to the trusted IP and threat lists, respectively. GuardDuty will not generate findings for IP addresses that are included in the trusted IP lists. This is done to avoid false alarms, especially from company IP addresses. However, we need to keep in mind that attacks can be internal too. Threat lists consist of known malicious IP addresses. The addresses we provide will be used along with the ones already available with AWS based on their research and experiences.

We can add an IP or a CIDR range to the trusted IP lists and threat lists in various formats, such as plain text (an IP or a CIDR per line), **Structured Threat Information Expression (STIX)**, **Open Threat Exchange (OTX)** CSV, FireEye iSIGHT Threat Intelligence CSV, Proofpoint ET Intelligence Feed CSV, and AlienVault Reputation Feed. Currently, we can have up to 2,000 lines in a trusted IP list and 250,000 lines in a threat list.

## There's more...

As a new feature, Amazon GuardDuty Malware Protection for S3 detects malicious file uploads to our selected Amazon S3 buckets. We can enable Malware Protection for S3 for a bucket that belongs to our own account and monitor the malware scan status using the embedded Amazon CloudWatch metrics in the GuardDuty console. We have seen this option in *Figure 8.1*.

Let's go through some important concepts related to GuardDuty:

- GuardDuty can detect compromised EC2 instances by analyzing VPC flow logs—for example, GuardDuty can detect whether an instance is used in a **Denial of Service (DOS)** attack.
- GuardDuty can detect whether our instances have been used for cryptocurrency mining. It can also detect whether our credentials have been stolen by accessing malicious IPs, using EC2 instance profiles outside of EC2, or making anomalous API calls to AWS resources such as S3, EC2, RDS, and so on.
- GuardDuty can scan for and detect malicious artifacts in EC2 instances and EKS infrastructure. It can also help identify malicious file uploads to S3 buckets.
- GuardDuty is a regional service. Even when multiple accounts are enabled and multiple AWS regions are used, the GuardDuty security findings remain in the same regions where the underlying data was generated.
- We can aggregate GuardDuty findings across different accounts into one account. We will discuss this in the *Aggregating findings from multiple accounts in GuardDuty* recipe.
- We can export GuardDuty findings across different accounts and regions to an Amazon S3 bucket to simplify the aggregation of all the findings. This is a new feature and is different from aggregating findings across different accounts into one account.
- We can use CloudWatch and SNS to monitor GuardDuty findings and send notifications.
- GuardDuty pricing is based on the amount of data analyzed:
  - For VPC flow log and DNS log analysis, GuardDuty charges us based on the size of the data analyzed.
  - For CloudTrail events, GuardDuty charges us based on the number of events analyzed.
- GuardDuty supports the processing, storage, and transmission of credit card data by a merchant or service provider, and has been validated as being compliant with the **Payment Card Industry (PCI) Data Security Standard (DSS)**

## Setting up Amazon EventBridge to monitor our findings from GuardDuty

The steps to set up **Amazon EventBridge** to monitor our findings from GuardDuty can be summarized as follows:

1. Go to the **EventBridge** service in the management console.

### Important note

EventBridge was formerly known as Amazon CloudWatch Events. As of now, if we navigate to CloudWatch and click on **Rules** under the **Events** section, we will be redirected to the EventBridge console.

2. Click on **Rules** under **Buses** from the left sidebar.
3. Click on **Create rule** to go to the **Create rule** page.
4. On the **Define rule detail** page, provide a name and description for our rule. Leave the value for **Event bus** as **default** and **Rule type** as **Rule with an event pattern** and click on **Next**.
5. In the **Build event pattern** section, scroll down to **Event pattern**.
6. Set **Event source** to **AWS events or Eventbridge partner events**.
7. Set **AWS service** to **GuardDuty**.
8. Set **Event Type** to **GuardDuty Finding**.
9. Click on **Next**. Under **Targets**, do the following:
  - I. Under **Select a target**, select **SNS topic** from the dropdown.
  - II. Select a topic. You can create an SNS topic by following the *Creating an SNS topic to send emails* recipe in *Chapter 7*.
10. Click on **Next**.
11. Optionally, click **Add new tags** and click on **Next**.
12. Review the rule and click on **Create rule**.
13. Navigate back to the **GuardDuty** service and click on **Settings** on the left sidebar.
14. Scroll down to **Sample findings** and click on **Generate sample findings**. After some time, check the inbox of the email address we configured for our SNS topic, and we should get an email about our GuardDuty findings.
15. To configure and modify the frequency with which GuardDuty updates Event Bridge and S3, we can go to the **GuardDuty** service, click on **Settings** from the left sidebar, and **Edit** in the **Findings export options** section.
16. If we have not yet configured the export to S3, we will be provided with a **Configure now** option to configure it.

## See also

Read more about GuardDuty at <https://www.cloudericks.com/blog/getting-started-with-amazon-guardduty>.

## Aggregating findings from multiple accounts in GuardDuty

In this recipe, we will configure GuardDuty to **aggregate findings** from multiple AWS accounts into a single account. Aggregating findings from multiple accounts into a single dedicated account provides a central place to query the findings from all our accounts. We can also make configuration changes in one place for all the accounts.

### Getting ready

We need the following to successfully complete this recipe:

- Two working AWS accounts are required. We will call them the aggregate account and the logger account. The aggregate account will aggregate logs from the member account.
- Note down the AWS account ID and the email address of the member account. If we are using AWS Organizations and IAM Identity Center, as we saw in *Chapter 1*, we can get these from the management account within the AWS Organizations service or the IAM Identity Center service.
- We should enable GuardDuty in the aggregate account and member by following the *Setting up and using Amazon GuardDuty* recipe from this chapter.

### How to do it...

We can configure GuardDuty to aggregate the findings from member accounts as follows:

1. Go to the **GuardDuty** service in the management console of the aggregate account.
2. Click on **Accounts** from the left sidebar.
3. Click on **Add accounts by invitation**.
4. Enter the AWS account ID and email address for the member account. Click on **Next**.
5. We should see our account on the **Accounts** page. The **Status** field will have the **Invite not sent** value.
6. Select the account we just added, click on the **Actions** dropdown, and click on **Invite**.
7. On the pop-up screen, you may optionally provide a message. Leave the selection for **Also send an email notification to the root user on the invitee's AWS account and generate an alert in the invitee's Personal Health Dashboard** unchecked and click on **Send invitation**. The status of our account should change to **Invited**.

8. Go to the **GuardDuty** service in the management console of the member account.
9. If we are enabling GuardDuty now, we will be taken to the **Invitations** page after enabling. Otherwise, go to the **Accounts** page from the left sidebar and click **Accept invitation**.
10. Click on **Settings** from the left sidebar to go to the **Settings** page and click on **Generate sample findings**.
11. Go to the **GuardDuty** dashboard in the aggregator account and check for findings from our member account.

In this recipe, we configured a member account to send logs to an aggregator account. The aggregate account will aggregate logs from the member account that we added and any other member accounts we will add later.

## How it works...

First, we logged in to the main account, which is the account that will aggregate all GuardDuty findings. We then invited a member account. We had already noted down the account ID, and so it should have known the root email address of this account. We then logged in to our member account and accepted the invitation. After that, we generated sample findings from the member account. Finally, we logged back into our main account and verified that the findings from the member account were now present in the main account.

## There's more...

GuardDuty retains the generated findings for 90 days. It exports the active findings to Amazon EventBridge (EventBridge). Additionally, we have the option to export these findings to an Amazon S3 bucket. This allows us to track historical data of potentially suspicious activities in our account and assess the effectiveness of the recommended remediation steps. We can do this by following these steps:

1. Configure the KMS key as follows:
  - I. **Create or choose a KMS key:** Select an existing KMS key or create a new one.
  - II. **Attach a policy to the KMS key:** Attach a policy to the KMS key to grant GuardDuty access. Refer to the `guardduty_kms_policy.json` file from the code files for sample policy content.
2. Configure the S3 bucket as follows:
  - I. **Choose an S3 bucket:** Either create a new bucket or use an existing one.
  - II. **Update the bucket policy:** Update the bucket policy to allow GuardDuty uploads. Refer to the `guardduty_s3_bucket_policy.json` file from the code files for sample policy content.

3. Configure findings export in GuardDuty Console as follows:
  - I. Got to the GuardDuty dashboard.
  - II. Navigate to **Settings** by clicking on the **Settings** option from the left sidebar.
  - III. Configure **Export Options** as follows:
    - i. Scroll down to the **Finding export options** section.
    - ii. Click on **Configure now**.
    - iii. Enter the S3 bucket and KMS key ARNs. Provide the ARN of the S3 bucket where the findings will be exported. Provide the ARN of the KMS key to encrypt the findings.
    - iv. Click on **Save** to apply the settings.

By following the preceding steps, we can configure AWS GuardDuty to export its findings to an S3 bucket, ensuring that the data is aggregated and secured across accounts and regions. Please make sure that you update the sample policy files with actual values as needed.

## See also

Read more about the export findings feature in GuardDuty at [https://docs.aws.amazon.com/guardduty/latest/ug/guardduty\\_exportfindings.html](https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_exportfindings.html).

## Setting up and using Amazon Macie

In this recipe, we will learn how to set up and use Amazon Macie. Macie is a machine learning-powered service in AWS and is used primarily to discover, classify, and protect sensitive data. Macie can analyze data in S3 buckets to find sensitive information, such as credentials, financial information, **Protected Health Information (PHI)**, **Personally Identifiable Information (PII)**, **API keys**, source code, and so on, and then classify them into different security levels. Macie can be used with CloudWatch to raise alarms and send notifications. Macie can also analyze API calls from CloudTrail to detect anomalies.

## Getting ready

We need the following to successfully complete this recipe:

- We will need a working AWS account and a user, as described in the *Technical requirements* section.
- We need an S3 bucket. The bucket should be in the same region in which we configure Macie. I have created a bucket in the `us-east-1` region.

- Our S3 bucket should have the following files with fake sensitive data and are provided with the code files associated with the chapter:
  - `credit-cards-data.txt` with sample credit card data
  - `custom-data-license-plates.txt` with sample license plates

## How to do it...

We can configure Macie to discover and classify risks in an S3 bucket as follows:

1. Go to the **Amazon Macie** service in the AWS management console.

### Important note

If we are using Macie for the first time, we should see a page with a **Get started** button. Click on **Get started**. Make sure that the region selected is the same as our bucket, view the service role permissions, and enable Macie to start a 30-day trial. We will then be taken to the **Summary** page. The 30-day free trial of automated sensitive data discovery does not cover sensitive data discovery jobs. If you initiate and execute a job, you will incur charges based on the total volume of uncompressed data analyzed by the job.

2. Click on **Jobs** in the left sidebar of the Macie dashboard.

### Important note

If we haven't configured a repository for our sensitive data discovery results, we will get a warning. We need to set up this repository within 30 days of enabling Macie. Please note that Macie only stores our sensitive data discovery results and findings for 90 days. We can click on **Configure** now to configure an existing or new bucket as the repository for sensitive data discovery results, or we can do it later (within 30 days) using the **Discovery results** menu item under **Settings** on the left sidebar.

3. Click on **Create job** to initiate the job creation process.
4. On the **Choose S3 buckets** page, opt for **Select specific buckets** to select the buckets you want to include in the job. We will select the bucket we created in the *Getting ready* section of this recipe.
5. Scroll down and click on **Next**.
6. For the **Review S3 buckets** page, review and verify our bucket selections, and then choose **Next**.
7. For the **Refine the scope** page, choose **One-time job**, provide the 100% value for **Sampling depth**, and click on **Next**.



**Important note**

Instead of choosing **One-time job**, we can select the **Scheduled job** option and set **Update frequency** to **Daily**, **Weekly**, or **Monthly**, but let us select **One-time job** for this recipe. Under **Advanced settings**, we can specify criteria to include or exclude certain objects from the job's analysis. If no criteria are provided, the job will analyze all objects in the buckets. For now, we should leave these settings as their defaults.

8. For **Managed data identifier options**, select **Recommended**.

## Select managed data identifiers [Info](#)

A managed data identifier is a set of built-in criteria that detects a specific type of sensitive data. Specify the types of sensitive data to detect by selecting managed data identifiers for the job to use.

### Managed data identifier options

A job can use multiple managed data identifiers. Specify which ones you want the job to use.

☒ **Recommended**  
Use all the managed data identifiers that AWS recommends for jobs.

☐ **Custom**  
Select specific managed data identifiers to use, or don't use any.

#### Recommended (35)

This table lists managed data identifiers that we recommend to detect common categories and types of sensitive data.

Sensitive data type ▾	Sensitive data category ▾
AUSTRALIA_TAX_FILE_NUMBER	PERSONAL_INFORMATION
AWS_CREDENTIALS	CREDENTIALS

Figure 8.5 – Managed data identifier options

9. Scroll down and click **Next**.
10. For **Custom data identifiers**, don't add anything and click **Next**.
11. For **Select allow lists**, don't add anything, and click **Next**.

12. On the **Enter general settings** pane, enter `AWS Sec CB Demo` under **Job name**. Optionally, provide a job description and add any tags as needed. Click **Next**.
13. On the **Review and create** page, take a moment to carefully examine the configuration settings to ensure their accuracy, and review the estimated total cost of the job. Click on **Submit**.
14. If we go to the **Jobs** page, the status of our job will be **Active (Running)**. Wait until the status is **Complete**. It could take 10 to 15 minutes.
15. Click on the hyper-linked job name from the list, and from the **Show results** dropdown, click on **Show findings** to view the findings. Click on the finding to learn more about it.

The screenshot shows the Amazon Macie Findings interface. At the top, it says "Amazon Macie > Findings" and "Showing 1 of 1". There are three severity filters: "Low: 0", "Medium: 0", and "High: 1". Below this, the "Findings (1)" section includes an "Info" link, a refresh button, and an "Actions" dropdown. A description states: "This table lists findings for your organization. Select a finding to show its details. You can also filter, group, and sort findings based on specific fields and field values." Below the description are buttons for "Suppress findings" and a "Saved rules" dropdown showing "No saved rules".

Under "Finding status", a dropdown menu is set to "Current". To the right, the "Filter criteria" section shows a filter rule: "Job ID: 4c076383bebbb9e4c0706c65f7537" with a value "ffa". There are buttons for "Add filter", "Save rule", and a close button. A pagination control shows "< 1 >".

At the bottom, a table lists the findings:

<input type="checkbox"/>	Severity ▾	Finding type ▾	Resources affected
<input type="checkbox"/>	High	SensitiveData:S3Object/Financial	awssecb/credit-cards-data.txt

Figure 8.6 – The Macie Findings tab

In this section, we saw how we can use Amazon Macie to find the S3 objects that contain financial information such as bank account numbers or credit card numbers. Next, we will see how we can use a custom Macie data identifier.

### ***Adding a custom Macie data identifier***

We can configure and use a custom Macie data identifier as follows:

1. Navigate back to the Macie service in the AWS Management Console.
2. Click on **Custom data identifiers** in the left sidebar.
3. Click on **Create**.

4. Provide `UKLicensePlates` under **Name**. For **Description - optional**, enter `UK License Plates`, and set the **Regular expression** as the following: `( [0-9] [a-zA-Z] [a-zA-Z] - ? [0-9] [a-zA-Z] [a-zA-Z] ) | ( [a-zA-Z] [a-zA-Z] [a-zA-Z] - ? [0-9] [0-9] [0-9] ) | ( [a-zA-Z] [a-zA-Z] - ? [0-9] [0-9] - ? [a-zA-Z] [a-zA-Z] ) | ( [0-9] [0-9] [0-9] - ? [a-zA-Z] [a-zA-Z] [a-zA-Z] ) | ( [0-9] [0-9] [0-9] - ? [0-9] [a-zA-Z] [a-zA-Z] )`
5. Leave everything else as default and click **Submit**.
6. Create a new job by following the steps in the previous section; however, on the **Custom data identifiers** page, select `UKLicensePlates`, which we created in this section.

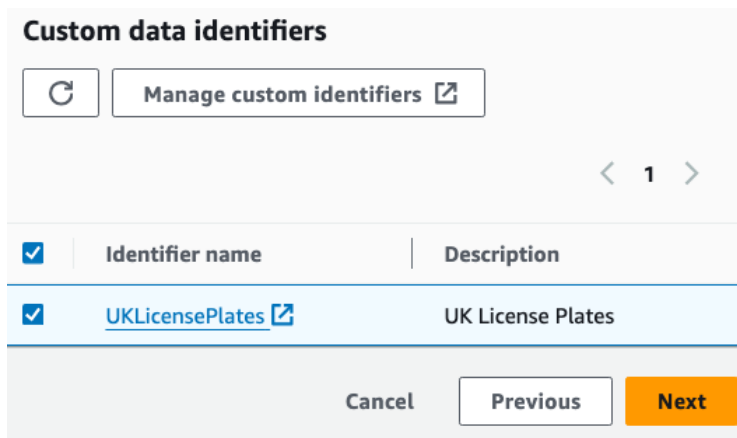


Figure 8.7 – Selecting custom data identifiers

7. Once the status of the job is **Complete** on the **Jobs** page, from the **Show results** dropdown, click on **Show findings**. We should be able to see a new finding corresponding to our custom identifier job.

## How it works...

Macie can be used to analyze S3 buckets and CloudTrail logs. In this recipe, we explored Amazon Macie, a machine learning-powered service in AWS that helps discover, classify, and protect sensitive data stored in S3 buckets. We set up Macie, initiated a discovery job to scan a specific S3 bucket, and observed how it successfully identified pre-defined sensitive data types such as credit card numbers. We then extended its capabilities by creating a custom data identifier named `UKLicensePlates` using a regular expression to recognize a specific format within our S3 bucket. This custom identifier was incorporated into a new discovery job. Upon completion, we confirmed that Macie successfully recognized the six license plates, showcasing its ability to discover both pre-defined and user-defined sensitive data types.

## There's more...

Let's go through some important concepts related to Macie:

- Macie can be used within enterprises for a variety of use cases. Macie can detect whether sensitive data or source code has been downloaded from unusual IP addresses. Macie can detect which users are causing the most high-risk events. Macie can group events by location and help us detect any activity from unknown locations. Macie can also give a high-level breakdown of the type of CloudTrail events within our account. If we see any unexpected calls, we can drill down to find out the root cause.
- Both Macie and GuardDuty have an overlap of functionalities related to analyzing API calls. Unlike GuardDuty, the focus of Macie is more on access patterns, such as uploading or downloading more data than is normally done. In general, it is preferable to use Macie alongside GuardDuty for better protection.
- Macie can be used alongside CloudWatch for raising alarms and sending notifications.
- Macie can aggregate data from multiple accounts. We can add additional accounts from the **ACCOUNTS** tab on the **INTEGRATIONS** page after setting up the required permissions as mentioned.
- Macie currently charges us for S3 content classification and CloudTrail event processing.
- Macie is a regional service. Macie must be enabled on a region-by-region basis and helps you view findings across all your accounts within each region.

The steps to configure notifications for Macie alerts with Amazon EventBridge and SNS can be summarized as follows:

1. Go to the **Amazon EventBridge** service in the console.
2. Click on **Rules** under **Buses** from the left sidebar.
3. Click on **Create rule** to go to the **Create rule** page.
4. On the **Define rule detail** page, provide a name and description for our rule. Leave **Event bus** and **Rule type** as **Rule with an event pattern** and click on **Next**.
5. In the **Build event pattern** section, scroll down to **Event pattern**.
6. Set **Event source** to **AWS services**.
7. Set **AWS service** to **Macie**.
8. Set **Event Type** to **Macie Finding**.
9. Click on **Next**.
10. In the **Targets** section, under **Select a target**, select **SNS topic** from the dropdown.

11. Select a **Topic**. You can create an SNS topic by following the *Creating an SNS topic to send emails* recipe in *Chapter 7*.
12. Click on **Next**.
13. Optionally, click **Add new tags** and click on **Next**.
14. Review the rule and click on **Create rule**.
15. Navigate back to **Macie service** and create a job for our S3 bucket by following the steps in the previous section.
16. After some time, check the inbox of the email we configured for our SNS topic, and we should get an email about our Macie findings.

By default, Macie automatically sends its findings to Amazon EventBridge for further processing and integration with other AWS services. You have the flexibility to configure Macie to deliver findings to additional destinations and define how frequently updates are sent to each destination by following these steps:

1. Navigate to the Macie console and click on **settings** on the left sidebar.
2. Scroll down to **Publication of findings** and update the frequency for the policy findings.

## See also

Read more about Amazon Macie at <https://www.cloudericks.com/blog/getting-started-with-amazon-macie>.

## Setting up and using Amazon Inspector

In this recipe, we will learn how to set up and use Amazon Inspector. Inspector is a service that performs automated **security assessments** to find vulnerabilities or deviations from standard practices for applications deployed on AWS. We can check the Inspector's findings directly in the console or from the detailed assessment report provided by the Inspector.

## Getting ready

We need the following to successfully complete this recipe:

- We need a working AWS account and a user, as described in the *Technical requirements* section.
- An EC2 instance in the default VPC, within a public subnet within the VPC, is also required. For **Amazon Machine Image (AMI)**, select **Amazon Linux 2023 AMI**. For **Instance type**, select **t2.micro**. For **Key pair (login)**, select an existing one you have access to or create a new one. Under **Network settings**, make sure that the value for **Auto-assign public IP** is **Enable**, and that **Create security group** is selected with the value for **Allow SSH traffic from** set to **Anywhere**.

## How to do it...

We can set up Inspector as follows:

1. Go to the **Amazon Inspector** service in the console.
2. If we are logging in for the first time, we should see a **Get started** page. Currently, there is a free 15-day trial for accounts that are new to Inspector. Click on **Get started**. On the **Activate Inspector** page, we should see a message that reads as follows: **When you activate Inspector, you grant Inspector permissions to discover, classify, and protect sensitive data in AWS on your behalf and to generate findings about potential security issues. This will activate Inspector for your account only.** Click on **Activate Inspector** and wait until it is fully activated. On the left sidebar, we should get options including the following:

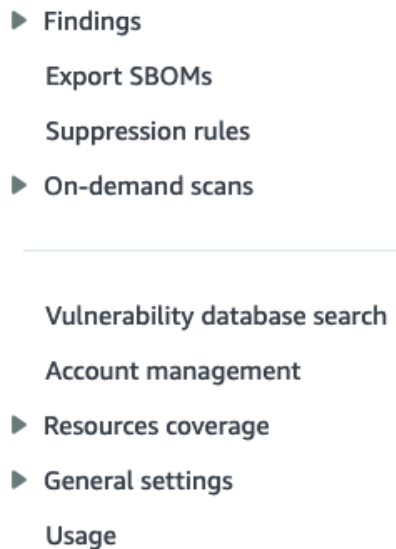
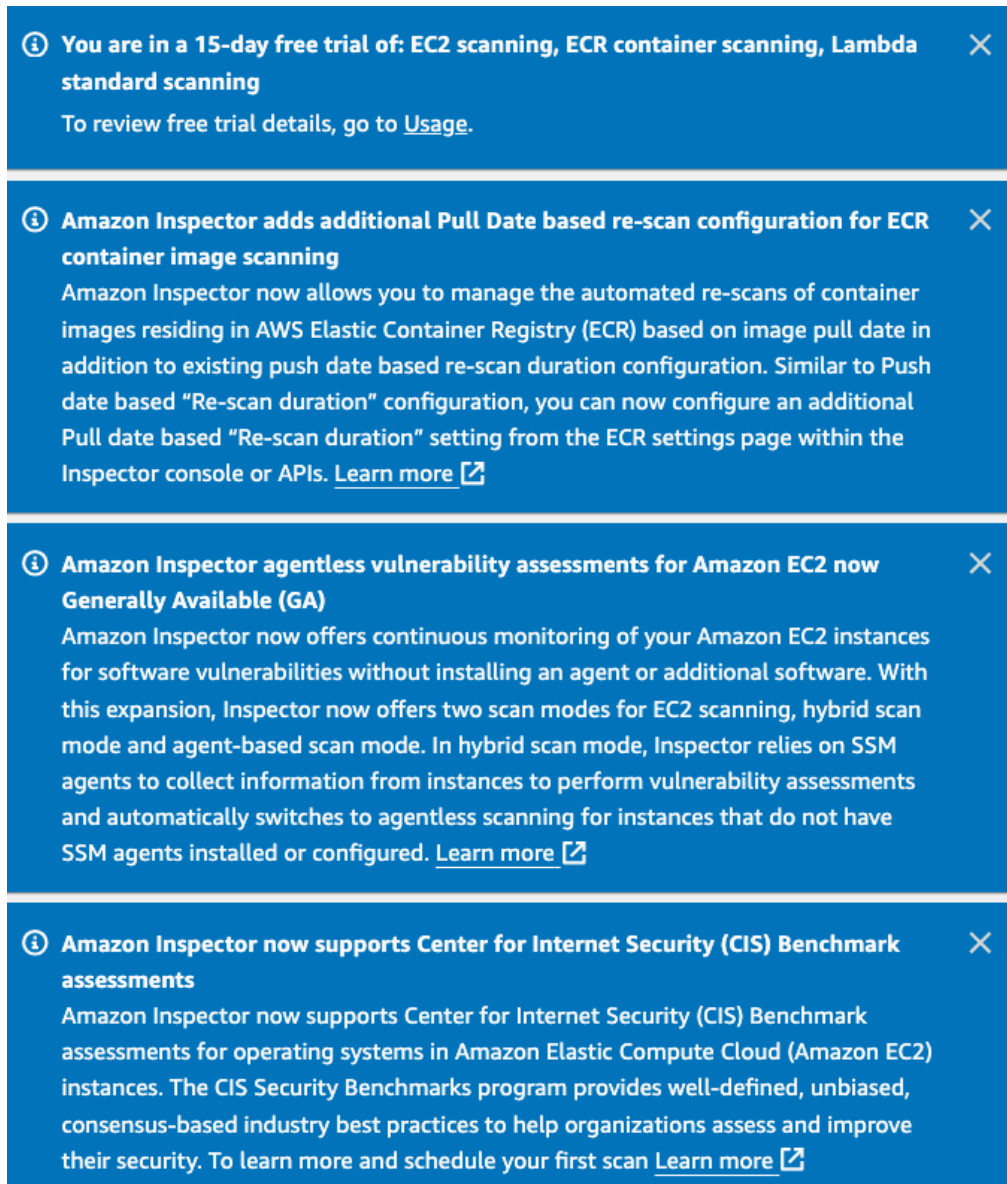


Figure 8.8 – Amazon Inspector’s left sidebar

If we expand **Findings**, we can see options such as **By vulnerability**, **By instance**, **By container image**, **By container repository**, **By Lambda function**, and **All findings**. We should also see messages such as the following, but it could be different in your case, as Amazon updates its user interface and messages frequently.



The image displays four stacked blue notification cards, each with a white information icon (i) on the left and a white close icon (X) on the right. The first card states: "You are in a 15-day free trial of: EC2 scanning, ECR container scanning, Lambda standard scanning. To review free trial details, go to [Usage](#)." The second card states: "Amazon Inspector adds additional Pull Date based re-scan configuration for ECR container image scanning. Amazon Inspector now allows you to manage the automated re-scans of container images residing in AWS Elastic Container Registry (ECR) based on image pull date in addition to existing push date based re-scan duration configuration. Similar to Push date based 'Re-scan duration' configuration, you can now configure an additional Pull date based 'Re-scan duration' setting from the ECR settings page within the Inspector console or APIs. [Learn more](#) [external link icon]." The third card states: "Amazon Inspector agentless vulnerability assessments for Amazon EC2 now Generally Available (GA). Amazon Inspector now offers continuous monitoring of your Amazon EC2 instances for software vulnerabilities without installing an agent or additional software. With this expansion, Inspector now offers two scan modes for EC2 scanning, hybrid scan mode and agent-based scan mode. In hybrid scan mode, Inspector relies on SSM agents to collect information from instances to perform vulnerability assessments and automatically switches to agentless scanning for instances that do not have SSM agents installed or configured. [Learn more](#) [external link icon]." The fourth card states: "Amazon Inspector now supports Center for Internet Security (CIS) Benchmark assessments. Amazon Inspector now supports Center for Internet Security (CIS) Benchmark assessments for operating systems in Amazon Elastic Compute Cloud (Amazon EC2) instances. The CIS Security Benchmarks program provides well-defined, unbiased, consensus-based industry best practices to help organizations assess and improve their security. To learn more and schedule your first scan [Learn more](#) [external link icon]."

**i** You are in a 15-day free trial of: EC2 scanning, ECR container scanning, Lambda standard scanning **X**  
To review free trial details, go to [Usage](#).

**i** Amazon Inspector adds additional Pull Date based re-scan configuration for ECR container image scanning **X**  
Amazon Inspector now allows you to manage the automated re-scans of container images residing in AWS Elastic Container Registry (ECR) based on image pull date in addition to existing push date based re-scan duration configuration. Similar to Push date based "Re-scan duration" configuration, you can now configure an additional Pull date based "Re-scan duration" setting from the ECR settings page within the Inspector console or APIs. [Learn more](#) [external link icon]

**i** Amazon Inspector agentless vulnerability assessments for Amazon EC2 now Generally Available (GA) **X**  
Amazon Inspector now offers continuous monitoring of your Amazon EC2 instances for software vulnerabilities without installing an agent or additional software. With this expansion, Inspector now offers two scan modes for EC2 scanning, hybrid scan mode and agent-based scan mode. In hybrid scan mode, Inspector relies on SSM agents to collect information from instances to perform vulnerability assessments and automatically switches to agentless scanning for instances that do not have SSM agents installed or configured. [Learn more](#) [external link icon]

**i** Amazon Inspector now supports Center for Internet Security (CIS) Benchmark assessments **X**  
Amazon Inspector now supports Center for Internet Security (CIS) Benchmark assessments for operating systems in Amazon Elastic Compute Cloud (Amazon EC2) instances. The CIS Security Benchmarks program provides well-defined, unbiased, consensus-based industry best practices to help organizations assess and improve their security. To learn more and schedule your first scan [Learn more](#) [external link icon]

Figure 8.9 – Messages post-activating Amazon Inspector

- Wait for some time, then click on **Account management** from the left sidebar, go to the **Instances** tab, and verify that the status of our instance is **Actively monitoring**. The **Monitored using** column will have a value of **Agentless** as we are using an EBS-backed EC2 instance.

4. In the left sidebar, expand **Findings** and click on **All findings**.

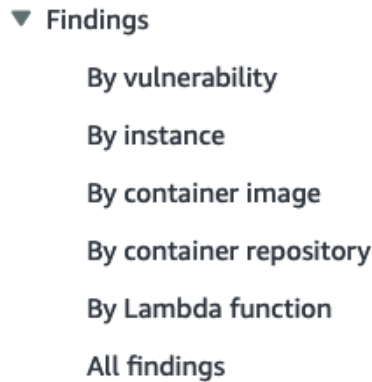


Figure 8.10 – Inspector's Findings menu

As we are using a newly launched EC2 instance with Amazon Linux, we won't see many findings, but we should see one with **Severity** as **Medium** for **Port 22 is reachable from an Internet Gateway – TCP**.

We quickly explored how to see the Inspector findings in this recipe. We will look into more options available within Inspector's dashboard in the *There's more...* section of this recipe.

## How it works...

Amazon Inspector for EC2 scanning extracts metadata from instances and compares it to security advisories to produce findings, focusing on package vulnerabilities and network reachability issues. Network reachability scans occur every 24 hours, while package vulnerability scans have variable frequencies based on the scan method. Amazon Inspector employs both agent-based and agentless scan methods, depending on the account's scan mode settings, to collect software inventory and detect vulnerabilities.

Agent-based scans use the SSM agent to continuously collect software inventory from eligible instances. These scans identify vulnerabilities in both operating systems and application programming language packages in Linux instances. To perform agent-based scans, instances must have a supported OS, be SSM managed, and not be excluded by specific tags. Amazon Inspector uses various SSM associations and plugins to gather inventory data and update vulnerability findings regularly. Additionally, agentless scans, part of the hybrid scanning mode, utilize EBS snapshots to collect inventory data from instances, scanning for operating system and application package vulnerabilities.



Amazon Inspector offers two scan modes: agent-based and hybrid scanning. The agent-based mode provides continuous scans for SSM-managed instances, while hybrid scanning combines both methods, using agent-based scans for SSM-managed instances and agentless scans for eligible EBS-backed instances. Users can configure scan modes, manage exclusions, and ensure proper SSM agent setup to optimize vulnerability detection and maintain effective security monitoring for their EC2 instances.

## There's more...

We explored how EC2 instances are scanned in this recipe. Amazon Inspector also supports scanning for the following resources:

- **Amazon Elastic Container Registry (ECR):** Amazon Inspector scans container images stored in Amazon ECR for vulnerabilities. This ensures that container images used in your environment are secure and free from known issues.
- **AWS Lambda:** Amazon Inspector scans AWS Lambda functions for vulnerabilities, checking the code and dependencies to ensure serverless applications remain secure.

In this recipe, we explored the **Findings** option in the left sidebar to review the findings. The left sidebar also offers several other options, including the following:

- **Export SBOMs:** The **Software Bill of Materials (SBOM)** provides a detailed list of software components and their relationships within a system. Amazon Inspector can generate and export SBOMs, which are helpful for understanding and managing software dependencies, ensuring compliance with security policies, and identifying potential vulnerabilities. This helps organizations maintain a comprehensive inventory of their software components.
- **Suppression rules:** Suppression rules in Amazon Inspector allow users to manage findings by suppressing specific types or sources of findings. This helps reduce noise and focus on critical vulnerabilities. Users can create suppression rules based on criteria such as vulnerability ID, resource tags, or resource types to ensure that only relevant findings are highlighted in security reports.
- **On-demand scans:** On-demand scans allow users to manually initiate scans for vulnerabilities or network reachability issues on EC2 instances as needed. This is particularly useful after deploying new software or making significant infrastructure changes. Under the **On-demand scans** menu, using the **CIS scans** option, we can evaluate the compliance of EC2 instances with **Center for Internet Security (CIS)** benchmarks, which are best practices for securing IT systems and data. These scans help organizations adhere to security standards and improve their security posture by identifying and addressing non-compliant configurations.

### **Note on Inspector Classic**

The Inspector dashboard currently has a sidebar link called **Switch to Inspector Classic**. Inspector Classic was the original version of Amazon Inspector, designed to help AWS users automate security assessments for their applications by checking for vulnerabilities and compliance issues. Although Inspector Classic was initially vital, it has largely been replaced by the new Amazon Inspector, which offers enhanced features and a more comprehensive, streamlined approach to security assessments.

Inspector Classic remains available for users with existing dependencies, but transitioning to the new Amazon Inspector is advisable for the latest features, improved performance, and enhanced security capabilities. AWS continues to support Inspector Classic for backward compatibility, but new users and assessments should use the updated Amazon Inspector for optimal security management. For those interested in working with Inspector Classic, refer to the first edition of this book or the link provided in the *See also* section.

### **See also**

- You can read more about Amazon Inspector at <https://www.cloudericks.com/blog/a-deep-dive-into-amazon-inspector-capabilities-and-integrations>.
- Read about Inspector Classic at <https://www.cloudericks.com/blog/understanding-inspector-classic-and-transitioning-to-the-new-amazon-inspector>.

## **Setting up and using AWS Security Hub**

In this recipe, we will explore how to set up and utilize AWS Security Hub. Security Hub aggregates findings from services such as Config, GuardDuty, Macie, and Inspector, offering a centralized platform to manage security alerts and automate compliance checks. Security Hub can do automated compliance checks using the CIS AWS Foundations Benchmarks, which is enabled by default when we enable Security Hub.

### **Getting ready**

We need the following to successfully complete this recipe:

- A working AWS account and a user, as described in the *Technical requirements* section.
- Set up AWS Config and enable recording as discussed in the *Setting up and using AWS Config* recipe in *Chapter 7*.
- Set up one or more of the following services: Amazon GuardDuty, Amazon Macie, and Amazon Inspector, following the respective recipes in *Chapters 7 and 8*.

## How to do it...

We can set up Security Hub in a region as follows:

1. Go to the **Security Hub** service in the AWS management console.
2. If we are using Security Hub for the first time, we should see a **Get started with Security Hub** section. Click on **Go to Security Hub**. In the **Enable AWS Config** section, we should see a message that AWS Config needs to be enabled with recording along with possible steps, which I assume you have done as mentioned in the *Getting ready* section.

### Enable AWS Config

Before you can enable Security Hub standards and controls, you must first enable resource recording in AWS Config. You must enable resource recording for all of the accounts and in all of the Regions where you plan to enable Security Hub standards and controls. If you do not first enable resource recording, you might experience problems when you enable Security Hub standards and controls. AWS Config bills separately for resource recording. For details, see the [AWS Config pricing page](#).

You can enable resource recording manually from the [AWS Config console](#), or you can choose Download to download and then deploy an AWS CloudFormation template as a StackSet. See our [documentation](#) for more details.

**Download**

Figure 8.11 – The Enable AWS Config message while enabling Security Hub

3. In the **Security standards** section, select the security standards we want to enable from the available options as we can see in *Figure 8.12*. In the **Delegated Administrator** section, we can add a delegated administrator account to manage Security Hub for this organization. Finally, click on **Enable Security Hub**.

### Security standards

Enabling AWS Security Hub grants it permissions to conduct security checks. [Service Linked Roles \(SLRs\)](#) with the following services are used to conduct security checks: Amazon CloudWatch, Amazon SNS, AWS Config, and AWS CloudTrail.

- ☒ Enable AWS Foundational Security Best Practices v1.0.0
- ☐ Enable AWS Resource Tagging Standard v1.0.0
- ☒ Enable CIS AWS Foundations Benchmark v1.2.0
- ☐ Enable CIS AWS Foundations Benchmark v1.4.0
- ☐ Enable CIS AWS Foundations Benchmark v3.0.0
- ☐ Enable NIST Special Publication 800-53 Revision 5
- ☐ Enable PCI DSS v3.2.1

Figure 8.12 – Setting security standards while enabling Security Hub

We should see a screen with sections for **Summary**, **Security standards**, **Insights**, and so on. It may take some time for the findings to get updated. So, wait for some time before proceeding with further steps.

4. Click on **Security standards** from the left sidebar. If we are following on from *Step 2*, for the **AWS Foundational Security Best Practices v1.0.0** and **CIS AWS Foundations Benchmark v1.2.0** standards, we should see the **View results** and **Disable standard** options. For the rest of the standards that are not enabled, we should see the **Enable Standard** option. If we are not following on from *Step 2*, we should enable the **AWS Foundational Security Best Practices v1.0.0** and **CIS AWS Foundations Benchmark v1.2.0** standards using the **Enable Standard** option and wait for some time until we see the **Security score** for each standard.

5. Once the **Security** score is available for the **AWS Foundational Security Best Practices v1.0.0**, click on **View results**. We should see the **Security score** and **Failed controls** values such as the following, but the exact details could differ from account to account.

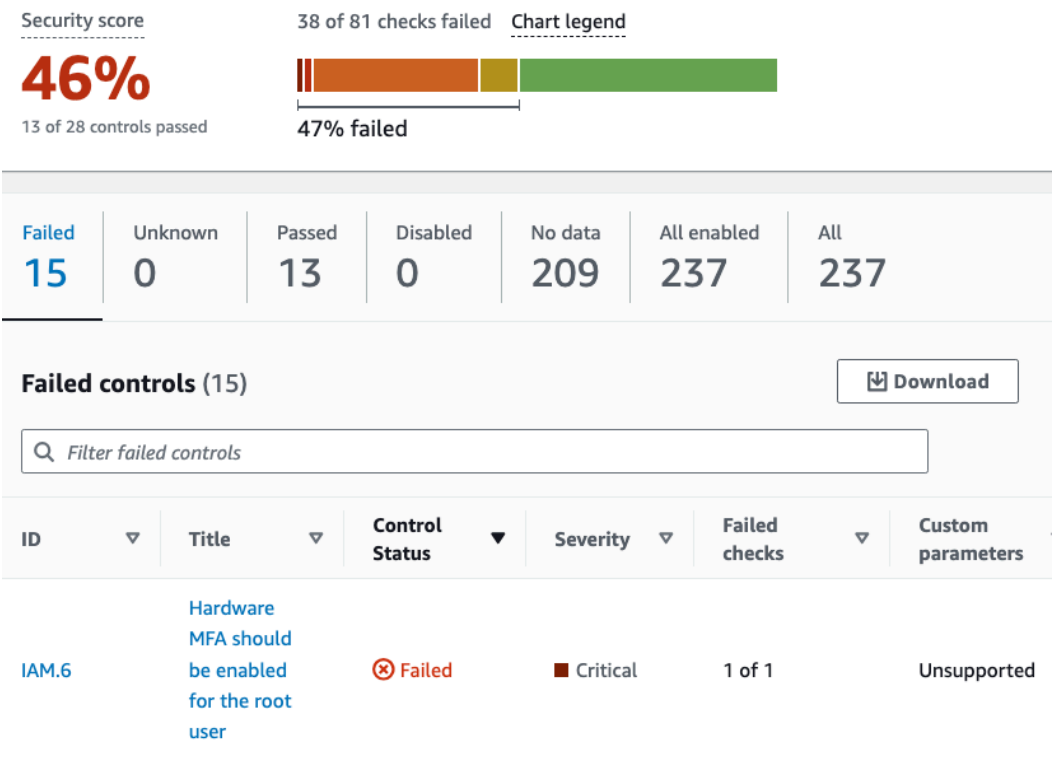


Figure 8.13 – Viewing the results for a security standard

6. Click on **Insights** on the left sidebar. We should see a list of existing insights, which are saved filters that display related findings. We can either select one of these existing insights or create a new one by clicking on the **Create insight** button.

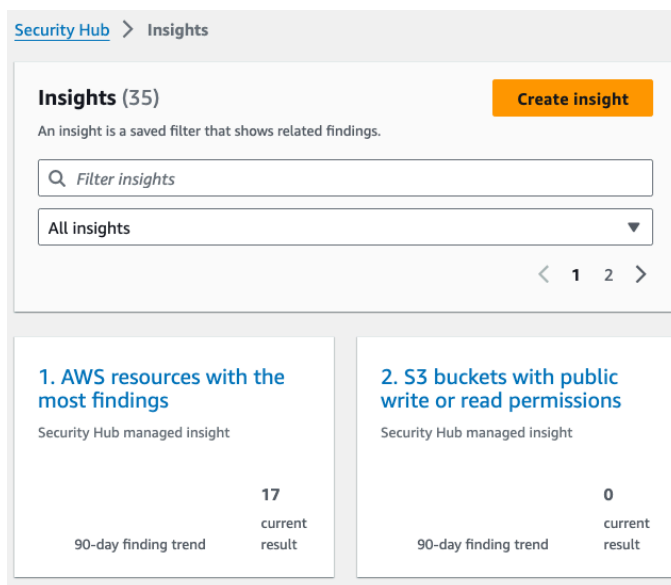


Figure 8.14 – Insights

- Click on **Findings** in the left sidebar to navigate to the **Findings** page. Here, you can view a list of findings from various service integrations. You can filter this list based on multiple criteria, such as the service that generated the finding or the severity of the finding.

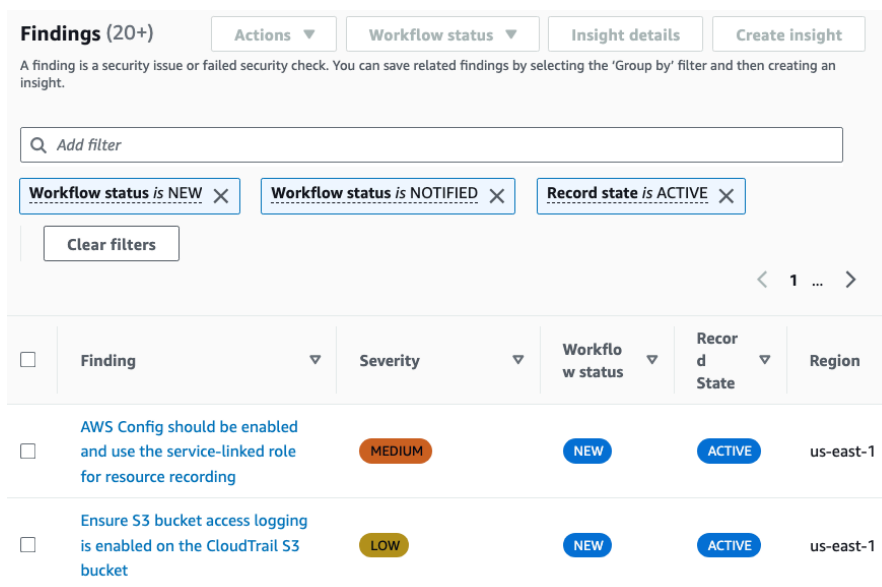


Figure 8.15 – Findings

8. Click on **Integrations** from the left sidebar to go to the **Integrations** page, where we can see the list of integrations currently enabled and those that are yet to be enabled. For integrations that are already enabled, we can disable them from the **Integrations** page.

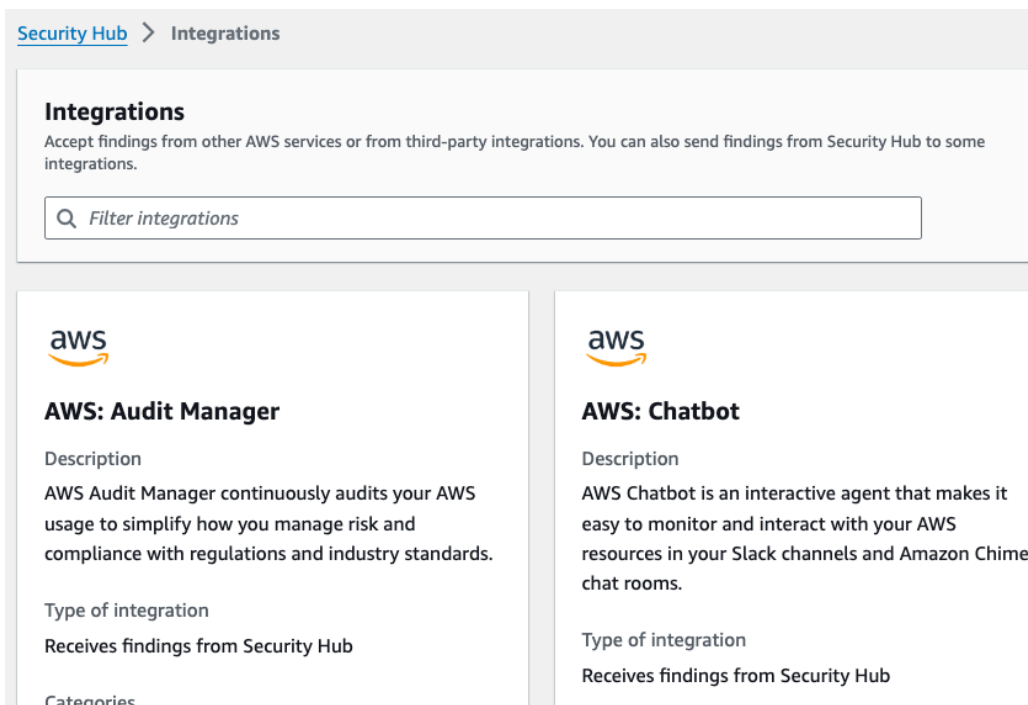


Figure 8.16 – Integrations

9. Click on **Usage** from the left sidebar to see the estimated usage and pricing, including **Usage during trial period**.
10. Go to the **Configuration** from the left sidebar.
11. If central configuration is not enabled, we will see a link to **Start central configuration** along with a warning message as follows: **Local configuration gives you limited access to security settings for your organization, for new accounts only. For additional configuration options, switch to central configuration. You can now use configuration policies to configure Security Hub across multiple Regions, accounts, and organizational units (OUs). When you switch, configuration policies replace the limited configuration settings for new organization accounts. We recommend central configuration for account management.**
12. In the **Accounts** section on the **Configuration** page, we can add additional member accounts to share their findings with this account. An invitation will be sent to the member accounts, and they must accept the invitation.

13. Click on the **Custom actions** from the left sidebar to send selected insights and findings to Amazon EventBridge.
14. Click on **Automations** from the left sidebar to update the Security Hub findings based on criteria that we define.
15. Click on **Regions** in the left sidebar to view findings across multiple regions. To start aggregating findings, click on the **Configure finding aggregation** button to set an aggregation region and then link other regions to it for a consolidated view of your security findings.

We can also disable Security Hub by clicking on **General** from the left sidebar and clicking on **Disable AWS Security Hub**. However, it is always recommended to have Security Hub enabled for all regions if the applicable costs are not a constraint.

## How it works...

AWS Security Hub acts as a centralized platform for security monitoring and management within the AWS environment. To get started, activate the service in the AWS Management Console. Once enabled, configure Security Hub to gather findings from a range of AWS services such as Amazon GuardDuty, AWS Inspector, and Amazon Macie, along with third-party tools and custom integrations.

The service provides continuous monitoring for security issues, collecting and organizing findings on a unified dashboard. It prioritizes these findings based on their severity and impact, helping users to address the most critical issues first. Within the Security Hub console, users can investigate these findings to gain insights into potential security threats and vulnerabilities. Additionally, Security Hub supports automated responses through AWS Lambda integration, allowing users to automate remediation actions for detected security issues.

In summary, AWS Security Hub offers a comprehensive solution for managing security posture, compliance, and incident response, thereby streamlining security operations and enhancing overall protection within the AWS environment.

## There's more...

Let's quickly go through some important concepts around Security Hub and its related services:

- Security Hub is a regional service. If cost is not a constraint, it is recommended to enable it in all regions.
- Security Hub can integrate with third-party security tools such as Alert Logic, Armor, Atlassian Opsgenie, and more.
- We can archive security findings from the **Findings** page so that older ones won't appear on the page.
- The CIS provides security standards for different servers, applications, and cloud providers. For example, they provide a set of security standards that are specific to AWS security.



- CIS Benchmarks for AWS can be categorized into four categories: identity and access management, logging, monitoring, and networking.
- IAM Access Analyzer uses logic-based reasoning to analyze resource-based policies in our AWS environment to inform us which resources in our account are shared with external principals.
- AWS Firewall Manager can be used to manage firewall rules across accounts and applications.

Learning about and understanding the CIS security benchmark controls for AWS will provide us with a better sense of security while working with AWS infrastructure. These controls can also help us make better decisions at work and even during exams.

## See also

- Read more about AWS Security Hub at <https://www.cloudericks.com/blog/getting-started-with-aws-security-hub>.
- You can find the list of CIS security standards supported by Security Hub at <https://docs.aws.amazon.com/securityhub/latest/userguide/securityhub-standards.html>.

## Using IAM Access Analyzer to inspect unused access

In this recipe, we will demonstrate how to utilize IAM Access Analyzer to identify and manage unused IAM resources within our environment. By focusing on detecting unused roles, access keys, and other critical components, we ensure adherence to security best practices. Through the process of configuring the analyzer to scan for unused resources, and interpreting the findings it generates, we gain valuable insights into potential security risks. By addressing any identified unused resources, we mitigate the risk of unauthorized access or misuse, enhancing the overall security posture of our IAM environment.

## Getting ready

We need the following to successfully complete this recipe:

- A working AWS account and a user, as described in the *Technical requirements* section.
- We need to have some IAM users and roles within our AWS account; we can create these by referring to the recipes in *Chapters 1 and 2*.

## How to do it...

We can use IAM Analyzer to inspect unused access as follows:

1. Navigate to the **IAM** service in the console.
2. Under **Access reports**, click on **Access Analyzer**.

3. Click on **Create analyzer**.
4. In the **Analysis** section, we can see options for **External access analysis** and **Unused access analysis**. Select the **Unused access analysis** option.

### Analysis

Select the type of finding to create an analyzer

Findings type | Info

☐ External access analysis  
The analyzer scans the resources within the zone of trust.

☒ Unused access analysis  
The analyzer scans IAM users and roles within selected organization or account.

IAM Access Analyzer charges for unused access analysis based on the number of IAM roles and users analyzed per analyzer per month. This includes creating analyzers across multiple Regions. For more details about pricing, see [IAM Access Analyzer pricing](#)

Figure 8.17 – Configuring the Findings type

5. In the **Analyzer details** section, leave the autogenerated **Name** for our Analyzer as-is. For **Zone of trust**, select **Current organization**.
6. Scroll down and click **Create analyzer**. Wait for some time for the findings to be populated.
7. Click on **Access Analyzer** in the left sidebar to see the **Findings overview** page that contains the findings summary.
8. Click on **Unused access** from the left sidebar. We should see findings related to unused access.

## How it works...

Using IAM Analyzer to inspect unused access works by employing it as a security scanner for your IAM resources. It analyzes roles, access keys, and permissions across your environment. By examining how recently these resources were used, it identifies elements that haven't been active for a specific period. This can indicate unnecessary access or forgotten configurations, potentially posing security vulnerabilities. With these insights, you can tighten access controls and improve the security posture of your IAM environment.

## There's more...

While identifying unused resources is a valuable capability, IAM Analyzer offers a broader range of functionalities to enhance your IAM security posture:

- **Verifying Permissions:** IAM Analyzer can validate policies against best practices and identify overly permissive settings that could grant unintended access.
- **Monitoring External Sharing:** Keep track of resources shared with external entities outside your AWS account. This helps identify potential security risks associated with unintended access granted to external users or applications.
- **Custom Policy Checks:** Define your own security standards and leverage IAM Analyzer to validate IAM policies against these custom checks. This empowers you to enforce specific security requirements within your organization.
- **Automating Policy Generation:** Simplify policy creation by leveraging IAM Analyzer's ability to generate IAM policies based on access activity logs from CloudTrail. This streamlines the process and helps ensure policies reflect actual usage patterns.
- **Continuous Monitoring:** IAM Analyzer is not a one-time scan; it provides ongoing monitoring of your IAM environment. This allows for the proactive identification of potential security issues as your IAM configuration evolves.

The steps to set up Amazon EventBridge to monitor our findings from Access Analyzer can be summarized as follows:

1. Go to the **Amazon EventBridge** service in the console.
2. Click on **Rules** under **Buses** from the left sidebar.
3. Click on **Create rule** to go to the **Create rule** page.
4. On the **Define rule detail** page, provide a name and description for our rule. Leave the **Event bus** as **default** and **Rule type** as **Rule with an event pattern** and click on **Next**.
5. In the **Build event pattern** section, scroll down to **Event pattern**.
6. Set **Event source** to **AWS services**.
7. Set **AWS service** to **Access Analyzer**.
8. Set **Event Type** to **Access Analyzer Finding**.
9. Set **Event Type Specification 1** to **Any resource by ARN**.
10. Click on **Next**. Under **Targets**, Choose **SNS topic** from the dropdown under **Select a target**.
11. Select a topic. You can create an SNS topic by following the *Creating an SNS topic to send emails* recipe in *Chapter 7*.

- 
12. Click on **Next**.
  13. Optionally, click **Add new tags** and click on **Next**.
  14. Review the rule and click on **Create rule**.

## See also

Read about essential tools to Secure IAM at <https://www.cloudericks.com/blog/essential-tools-to-secure-iam>.



# Advanced Identity and Directory Management

Having established a foundation in **IAM Identity Center** (formerly **AWS SSO**), IAM users, IAM groups, IAM roles, and IAM policies for identity and access management in previous chapters, we are now set to explore the more advanced aspects of user identity and directory management.

We will first delve into serverless identity-as-a-service user management strategies within AWS using **Amazon Cognito**. We will explore the two key functionalities of Cognito within this chapter: **user pools** and **identity pools**. User pools act as robust, scalable directories that handle user registration, authentication, and integration with our applications, along with features such as **multi-factor authentication (MFA)** and **federated identity** logins. Conversely, identity pools allocate temporary AWS credentials to authenticated users, allowing them to use AWS services such as Amazon S3, Amazon DynamoDB, and AWS Lambda.

Next, we will delve into the **directory service** solutions available within AWS, including AWS Simple AD, AWS Active Directory, and AD Connect. We will also examine the integration of **Microsoft Entra ID** (previously known as **Azure Active Directory**) with IAM Identity Center, which allows Entra ID users to access AWS resources effortlessly.

## Important note

This chapter's recipes might require supplementary knowledge, including familiarity with Microsoft products such as Active Directory and Microsoft Entra ID, as well as AWS services and features that have not yet been covered in this book. The recipes within this chapter are not mandatory for understanding and practicing subsequent recipes within the book. Therefore, if you are progressing through the chapters in this book and lack knowledge of the required products or services, and if you currently have no need for these solutions, feel free to read the steps within the recipes for understanding and skip practicing them until they are needed, and you have the required prerequisite knowledge. You may also learn enough about the required products and services from the links provided within the respective recipes.

This chapter contains the following recipes:

- Working with Amazon Cognito user pools
- Using identity pools to access AWS resources
- Using AWS Simple AD for creating a lightweight directory solution
- Using Microsoft Entra ID as the identity provider within AWS

## Technical requirements

Before diving into the recipes of this chapter, we need to ensure we have the following requirements and knowledge in place:

- We need an active AWS account to complete the recipes within this chapter. If we are using AWS Organizations, we can use the management account of the Organization, as we will be configuring many things at the AWS Organization level in this chapter. I will be using the `aws-sec-cookbook-1` account that we created in the *Multi-account management with AWS Organizations* recipe in *Chapter 1*.
- For administrative actions, we need a user who has **AdministratorAccess** permission to the AWS account we are working with.

Code files for this book are available at <https://github.com/PacktPublishing/AWS-Security-Cookbook-Second-Edition>. The code files for this chapter are available at <https://github.com/PacktPublishing/AWS-Security-Cookbook-Second-Edition/tree/main/Chapter09>.

## Working with Amazon Cognito user pools

Amazon Cognito is used primarily for two use cases: secure user identity management for our applications using its user pools feature and secure access to AWS resources by making use of temporary credentials using its identity pools feature. In this recipe, we will explore the user pools feature of Amazon Cognito by creating a user pool from the AWS Management Console and then creating a user within it.

## Getting ready

To complete this recipe, we need Amazon **Simple Notification Service (SNS)** for sending SMS if we plan to use SMS verification within this recipe.

### Important note

When we start using Amazon SNS for SMS messaging, our AWS account operates in an SMS sandbox. This sandbox serves as a secure space to test Amazon SNS functionalities without compromising our sender reputation. While in the sandbox, we are limited to sending SMS messages to verified destination phone numbers only.

If our AWS account is within the SMS sandbox, and we want to use SMS verification within this recipe, we can add a phone number to **Sandbox destination phone numbers** as follows:

1. Log in to the AWS Management Console and navigate to the Amazon SNS service.
2. Click on **Text messaging (SMS)** from the left sidebar.
3. Verify under **Account information** that this account is in the SMS sandbox.
4. In the **Sandbox destination phone numbers** section, click on **Add phone number**.
5. On the **Add a phone number** page, enter the phone number, selecting the correct country code. Also, choose the verification message language, which is the language the verification message will be sent in.
6. Click on **Add phone number**. This will take us to the **Verify phone number** page.
7. On the **Verify phone number** page, enter the verification code received on the provided phone number and click **Verify phone number**.

If it is successfully verified, the phone number should now appear with the **Verification status** as **Verified** in the **Sandbox destination phone numbers** section on the **Text messaging (SMS)** page.

We can explore more about Amazon SNS using the link provided in the *See also* section of this recipe. Assuming that we have set the environment up for this recipe as discussed within this section, we can get started with creating a Cognito user pool.



## How to do it...

We can create a Cognito user pool from the AWS Management Console as follows:

1. Log in to the AWS Account Management Console and navigate to the Cognito service. We should see the option to get started based on the business case. The default is **Add user directories to your app**, which is done using user pools.

### Start from your business case

Add user directories to your app ▼

Amazon Cognito user pools are a managed service that lets you add secure authentication and authorization to your apps, and can scale to support millions of users.

Create user pool

Figure 9.1 – Creating a user pool business case

The drop-down menu also includes the **Grant access to AWS services** option for business cases using identity pools.

2. Select the business case for **Add user directories to your app**, as shown in *Figure 9.1*, and click on **Create user pool**. We can also create user pools by going to the **User pools** page by clicking on the **User pools** menu option from the left sidebar.
3. Under **Provider types**, select only **Cognito user pool** (which is the default). Under **Cognito user pool sign-in options**, select **User name**, **Email**, and **Phone number**. Leave the other options as-is and click **Next**.

## Authentication providers

Configure the providers that are available to users when they sign in.

### Provider types

Choose whether users will sign in to your Cognito user pool, a federated identity provider, or both. Amazon Cognito has different pricing for federated users and user pool users. [Learn more about pricing](#)

#### ☒ Cognito user pool

Users can sign in using their email address, phone number, or user name. User attributes, group memberships, and security settings will be stored and configured in your user pool.

#### ☐ Federated identity providers

Users can sign in using credentials from social identity providers like Facebook, Google, Amazon, and Apple; or using credentials from external directories through SAML or Open ID Connect. You can manage user attribute mappings and security for federated users in your user pool.


### Cognito user pool sign-in options [Info](#)

Choose the attributes in your user pool that are used to sign in. If you select only one attribute, or you select a user name and at least one other attribute, your user can sign in with all of the selected options. If you select only phone number and email, your user will be prompted to select one of the two sign-in options when they sign up.

- ☒ User name
- ☒ Email
- ☒ Phone number

#### User name requirements

- ☐ Allow users to sign in with a preferred user name
- ☐ Make user name case sensitive

 Cognito user pool sign-in options can't be changed after the user pool has been created.

Cancel

Next

Figure 9.2 – Configuring authentication providers

4. On the **Configure security requirements** page, set **Password policy mode** to **Cognito defaults** (which is the default).

5. For **Multi-factor authentication**, select **Require MFA - Recommended** select **Authenticator apps**.

**Multi-factor authentication**

Configure secure access to your app by enforcing multi-factor authentication (MFA) during the user sign-in process. MFA settings are applied to all app clients.

MFA enforcement [Info](#)

<input checked="" type="radio"/> <b>Require MFA - Recommended</b> Users must provide an additional authentication factor when signing in.	<input type="radio"/> <b>Optional MFA</b> Users can sign in with a single authentication factor, and can choose to add additional authentication factors.	<input type="radio"/> <b>No MFA</b> Users can only sign in with a single authentication factor. This is the least secure option.
--	--	---

MFA methods [Info](#)

Choose the MFA methods that are allowed in your user pool. TOTP-based MFA offers a higher level of security. Recipient message and data rates apply.

☒ **Authenticator apps**  
Users can authenticate with a TOTP from an authenticator app such as Authy or Google Authenticator.

☐ **SMS message**  
Users can authenticate with a code sent by SMS message to a verified phone number. SMS messages are charged separately by Amazon SNS. [Learn more about pricing](#)

Figure 9.3 – Configuring the MFA settings during user pool creation

6. In the **User account recovery** section, select **Enable self-service account recovery – Recommended**. Under **Delivery method for user account recovery messages**, select **Email only**. Then click **Next**.

## User account recovery

Configure how users will recover their account when they forget their password. Recipient message and data rates apply.

### Self-service account recovery [Info](#)

☒ **Enable self-service account recovery - Recommended**

Allow forgot-password operations in your user pool. In the hosted UI sign-in page, a "Forgot your password?" link is displayed. When this feature is not enabled, administrators reset passwords with the Cognito API.

### Delivery method for user account recovery messages [Info](#)

Select how your user pool will deliver messages when users request an account recovery code. SMS messages are charged separately by Amazon SNS. Email messages are charged separately by Amazon SES. [Learn more about pricing.](#)

☒ **Email only**

☐ SMS only

☐ Email if available, otherwise SMS

☐ SMS if available, otherwise email

☐ SMS if available, otherwise email, and allow a user to reset their password via SMS if they are also using it for MFA

Cancel

Previous

Next

Figure 9.4 – Configuring the user account recovery settings during user pool creation

7. On the **Configure sign-up experience** page, select **Enable self-registration**, **Allow Cognito to automatically send messages to verify and confirm - Recommended**, and **Send email message, verify email address**. Leave the other values as-is and click **Next**.

### Important note

Enabling user sign-up in our user pool allows individuals from anywhere on the internet to register for an account and log in to our applications. We need to avoid turning it on unless we are ready to allow public sign-up access to our app. We can also personalize the sign-up experience by incorporating up to 50 custom attributes from the sign-up experience page. However, it's important to note that once a user pool is established, the names of these custom attributes cannot be modified.

8. On the **Configure message delivery** page, select **Send email with Cognito**.

### Important note

We can initially utilize Cognito's default email address for development purposes, which supports up to 50 emails daily. If we have established a verified sender through Amazon SES and wish to utilize its features, we should choose the **Send email with Amazon SES - Recommended** option and input the necessary SES details.

9. Provide the `SecCbCognitoUserPoolRole` value for **IAM role name** within **SMS settings**. Leave everything else as-is and click **Next**.

## SMS

Configure how your user pool sends SMS messages to your users. Recipient message and data rates apply.

### IAM role [Info](#)

Choose an IAM role for Cognito to use to send SMS messages with Amazon SNS.

- ☒ **Create a new IAM role**  
Cognito can create a role for you with the permissions required to send SMS messages.

- ☐ **Use an existing IAM role**  
Your existing role must have a trust relationship with `cognito-idp.amazonaws.com` and have at least `sns:Publish` permissions.

### IAM role name

Enter a name for your new IAM role.

`SecCbCognitoUserPoolRole`

Role names may be up to 64 characters long and can use alphanumeric characters, as well as the following special characters: `+ = , @ - _`

### SNS Region [Info](#)

US East (N. Virginia)



#### Configure AWS service dependencies to complete your SMS message setup

To send SMS messages from this user pool, you must complete the following additional steps if you haven't already done so. If you do not complete this now, we will remind you later. [Learn more](#)

The service links that follow may redirect you to a different AWS Region.

- ▶ [Request an Amazon SNS spending limit increase](#)
- ▶ [Move to Amazon SNS production environment](#)
- ▶ [Set up an Amazon Pinpoint originating identity](#)

Cancel

Previous

Next

Figure 9.5 – The SMS settings during user pool creation

### Important note

If our AWS account is currently in the SMS sandbox, we must add and verify any phone numbers we intend to use for SMS verification to the sandbox's list of verified destination phone numbers, as outlined in the *Getting ready* section of this recipe. If our account is not in the sandbox, we need to configure SMS messaging by following the instructions provided by Amazon on the screen, as shown in *Figure 9.5*. Please note that a detailed walkthrough of setting up SMS messaging is beyond the scope of this book.

10. On the **Integrate your app** page, select **User pool name** under **SecCbUserPool**, and select **Use the Cognito Hosted UI**. In the **Domain** section, select **Use a Cognito domain** and enter a unique domain prefix.

#### Important note

We can also input a custom domain that we own to be used for Cognito-hosted registration and login pages. To utilize a custom domain, providing a DNS record and a certificate from **AWS Certificate Manager (ACM)** is a prerequisite. For production workloads, AWS recommends the use of a custom domain to enhance professionalism and branding.

11. In the **Initial app client** section, provide `Sec Cb App` under **App client name**. Under **Client secret**, select **Generate a client secret**, and for **Allowed callback URLs**, give a URL to redirect after authentication, say `https://www.cloudericks.com`. Click **Next**.

#### Important note

Client secrets serve as a means for the server-side part of an app to authenticate API requests, offering a layer of security that helps prevent third parties from impersonating your client. It is essential to note that once Amazon Cognito generates a client secret for your app client, it cannot be modified or deleted.

12. Review all details and click **Create user pools**. We should now be able to see our new user pool on the **User pools** page.
13. From the **User pools** page, click on the hyperlinked name of the user pool we created to access its settings page. The settings page should have a **User pool overview** section and the **Delete user pool** button, as shown in the following figure:

**SecCbUserPool** [Info](#) Delete user pool

#### User pool overview

User pool name SecCbUserPool	ARN <code>arn:aws:cognito-idp:us-east-1:207849759248:userpool/us-east-1_EtNDsT79</code>	Created time October 20, 2023 at 14:15 GMT+5:30
User pool ID <code>us-east-1_EtNDsT79</code>	Estimated number of users 0	Last updated time October 20, 2023 at 14:15 GMT+5:30

Figure 9.6 – The User pool overview page

14. Scroll down and navigate to the **Users** tab, which is the first tab.

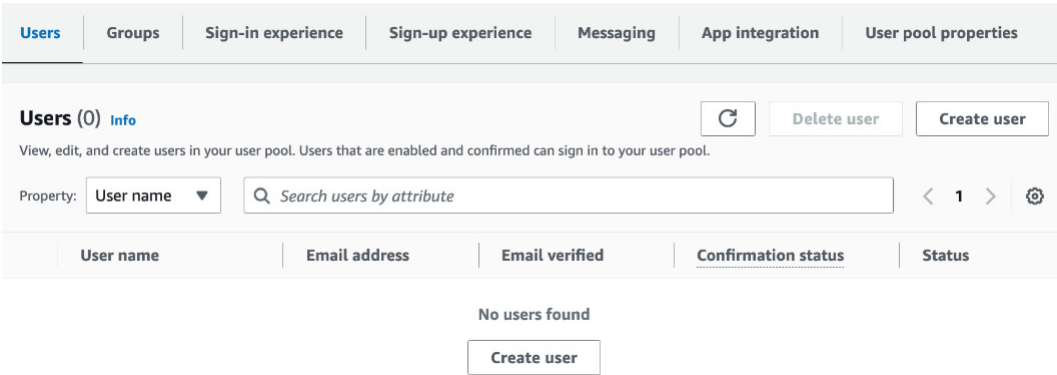


Figure 9.7 – The Users tab for the user pool

15. Click on **Create user**.

16. On the **Create user** page, select **Email** and **Phone** under **Alias attributes used to sign in**.

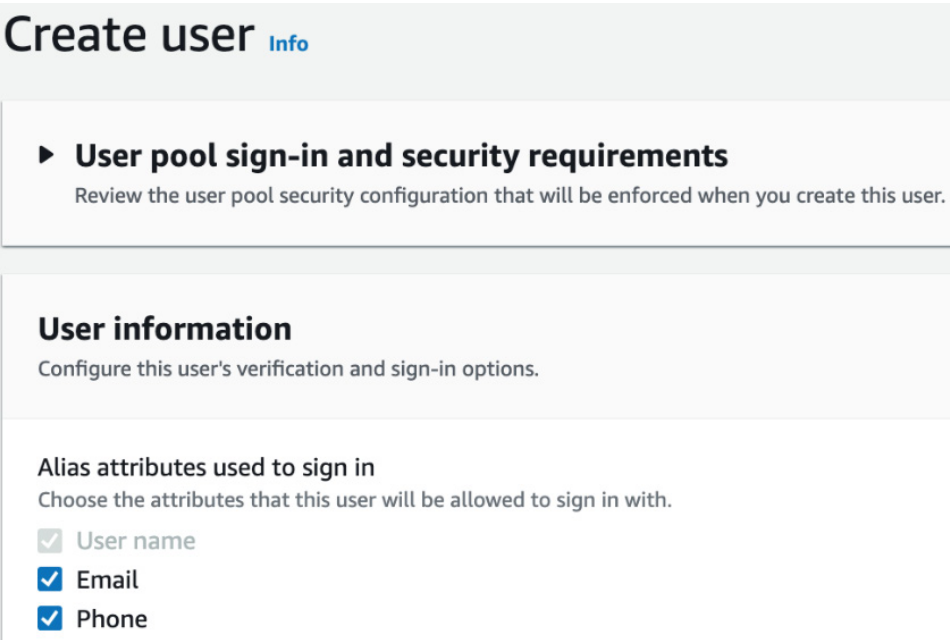



Figure 9.8 – Configuring the alias attributes used to sign in

17. Select **Don't send an invitation** for the **Invitation message** option.

**Invitation message** [Info](#)  
Configure invitation message templates in the [Messaging tab](#) 

☒ Don't send an invitation

☐ Send an email invitation

☐ Send an SMS invitation

☐ Send both email and phone SMS invitation

Figure 9.9 – Configuring the invitation message

18. Provide **User name**, **Email address**, **Phone number**, and **Temporary password** for the user, and click on **Create user**.

**User name**  
User name is an required attribute based on your user pool and above configurations.

heartin

**Email address**  
Enter this user's email address. A user's email address can be used for sign-in, account recovery, and account confirmation. The user must confirm ownership of this email address before they can sign in.

heartin@cloudericks.com

☒ Mark email address as verified

**Phone number**  
Enter the user's phone number, including country code. The phone number is a required attribute for sign-in based on your selections and user pool configuration. The user must confirm ownership of this phone number before they can sign in.

+919094403333

☒ Mark phone number as verified

**Temporary password**  
Amazon Cognito will send the password you generate to the user in an email message.

☒ Set a password

☐ Generate a password

**Password**  
Enter a temporary password for this user. The temporary password will be sent to the user in their invitation message.

••••••••

☐ Show password

Cancel Create user

Figure 9.10 – New user configuration



The new user should now appear in the **Users** tab similar to the following figure:

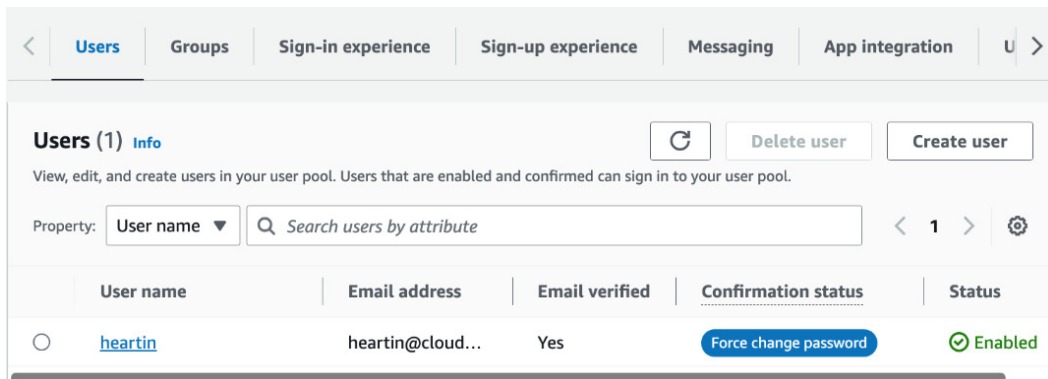


Figure 9.11 – The Users tab with the new user

19. From the user pool's settings page, navigate to the **App integration** tab and click on the hyperlinked **App client name** of the initial app client that was created as part of this recipe. We should now see our app's settings page.
20. In the **Hosted UI** section within the app's settings page, click on **View Hosted UI**. We should now see the default Hosted UI sign-in page, as follows:

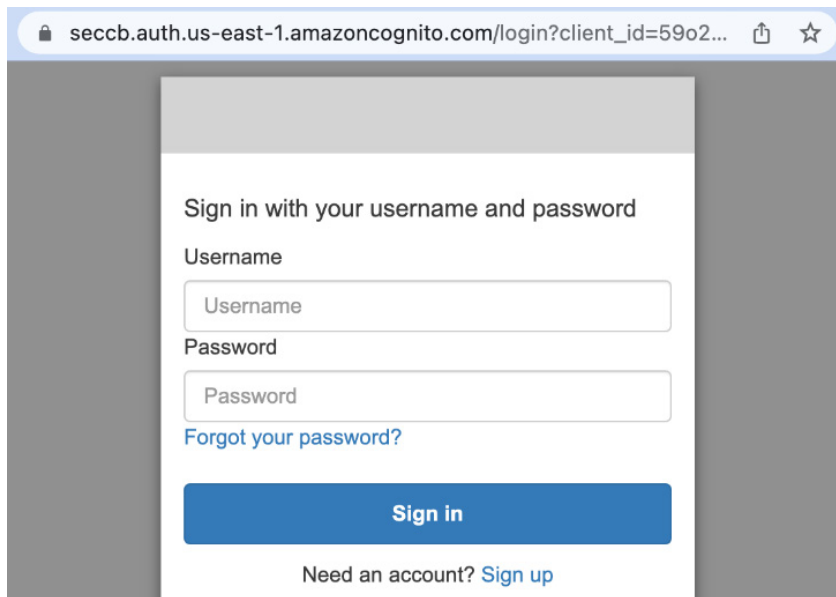


Figure 9.12 – The hosted UI sign-in page

21. Enter the username and password for the user we created in this recipe and click on **Sign in**. We will be taken to the **Change Password** screen.
22. Within the **Change Password** screen, enter a new password under the checkboxes with the **New Password** and **Enter New Password Again text** labels, and click on **Send**. This will send a **One Time Password (OTP)** to the phone number provided while creating the user and ask us to enter it for verification, as shown in the following figure:

We have delivered the authentication code by SMS to +\*\*\*\*\*3333. Please enter the code to complete authentication.

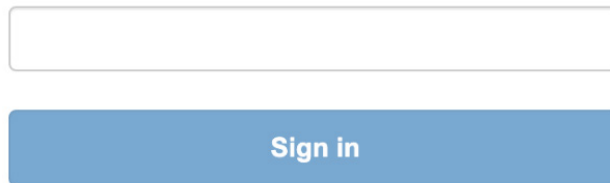
A screenshot of a web form for SMS verification. It features a single-line text input field with a light gray border. Below the input field is a solid blue rectangular button with the text "Sign in" in white, centered.

Figure 9.13 – SMS verification

Once verified, we will be redirected to the configured URL. If you were following the recipe and provided the `https://www.cloudericks.com` URL, you should see a screen similar to the following:

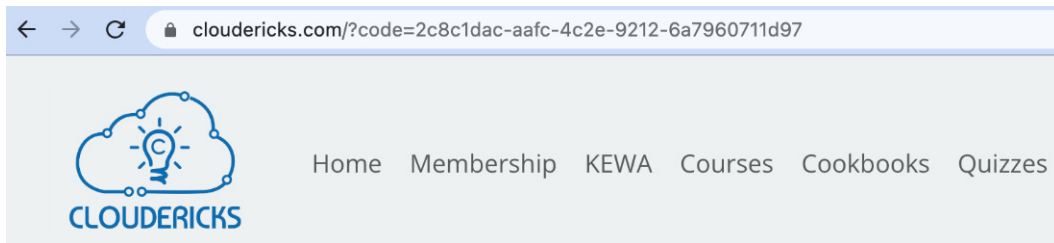


Figure 9.14 – The post-sign in page

In this recipe, we created a user pool and then created a user from the AWS Management Console. Alternatively, users can sign up on their own through the hosted UI's sign-up option. Customizations for the sign-in and sign-up experiences, and configuring the **Groups**, **Messaging**, **App integration**, and **User pool** properties, can be made from the user pool's settings page by navigating to their respective tabs, as shown in *Figure 9.7*.

## How it works...

Amazon Cognito provides two primary features: user pools and identity pools. User pools serve as secure directories that handle sign-up and sign-in operations for app users, while identity pools are mechanisms for issuing AWS credentials to authenticated users, enabling access to the AWS services. In this recipe, we created a user pool.

For the user pool sign-in options, we selected **Username, Email, and Phone number**. We can have one or both email or phone number as the username and then use these to sign in instead of username. For enhanced security, we also selected MFA and specified the account recovery mechanisms. For a better sign-up experience, we enabled self-registration and configured automatic message verification.

We used the **Cognito domain** option for Cognito-hosted registration and login pages to keep the recipe simple. We can also use a custom domain that we own instead. To utilize a custom domain, providing a DNS record and a certificate from ACM is a prerequisite. For production workloads, AWS recommends the use of a custom domain to enhance professionalism and branding.

No custom attributes were set up during our process. However, the sign-up experience can be tailored by introducing up to 50 custom attributes. Once a user pool is established, it's important to remember that the names of these custom attributes become fixed and cannot be altered. We can, however, add new ones later.

An initial app client was also created during the process with default settings. Amazon Cognito client apps work by authenticating users and then exchanging tokens with Amazon Cognito for temporary AWS credentials. These temporary AWS credentials can then be used to access AWS resources.

Client apps within Cognito can be classified into three categories:

- **Public client:** This is typically a client-side application such as a native app on a mobile device or a browser-based app. In this scenario, the app makes API requests from devices that cannot be entrusted with a client secret due to the potential for exposure.
- **Confidential client:** This is often a server-side application that has the capability to securely store a client secret. API requests to Cognito are routed through a central server, which is considered secure and capable of protecting this sensitive information.
- **Custom app:** This option allows for a more tailored setup. You have the autonomy to define the specific grant types, authentication flows, and whether a client secret is required based on your unique requirements and security considerations.

We will look into some more details including alternative business cases in the next section.

## There's more...

In this recipe, we created an Amazon Cognito user pool by selecting the **Add user directories to your app** business case, as shown in *Figure 9.1*. We can instead select the **Grant access to AWS services** business case from the dropdown to create an identity pool, as shown in the following figure:

### Start from your business case

Grant access to AWS services ▼

Cognito identity pools let you get temporary credentials to access AWS services for signed-in users as well as anonymous guest users.

Create identity pool

Figure 9.15 – Creating an identity pool business case

In the recipe, under **Provider types**, we only selected the Cognito user pool (which is the default), as shown in *Figure 9.2*. As shown in that figure, the **Cognito user pool** option is selected by default and cannot be unchecked, and we also have an option called **Federated identity providers**. If we select the **Federated identity providers** option, we will get additional sign-in options enabling us to use credentials from popular social IdPs such as Facebook, Google, Amazon, and Apple, or from external directories via SAML or **Open ID Connect (OIDC)** protocols, as shown in the following figure:

### Federated sign-in options [Info](#)

Choose the third-party identity providers that you would like to configure. You can add or remove federation providers at any time.

<input type="checkbox"/> <b>Facebook</b> Allow sign-in with a Facebook account.	<input type="checkbox"/> <b>Google</b> Allow sign-in with a Google account.	<input type="checkbox"/> <b>Login with Amazon</b> Allow sign-in with an Amazon account.
<input type="checkbox"/> <b>Sign in with Apple</b> Allow sign-in with an Apple ID.	<input type="checkbox"/> <b>SAML</b> Configure trust between Cognito and a SAML 2.0-compatible identity provider, such as Okta or Active Directory Federation Services.	<input type="checkbox"/> <b>OpenID Connect (OIDC)</b> Configure trust between Cognito and an OIDC identity provider, such as PayPal or Salesforce.

Figure 9.16 – Federated sign-in options for user pools

Selecting the **Cognito user pool** option is mandatory since it maintains profiles for both direct and federated users within our user pool. However, if our goal is to exclusively offer federated sign-in options, we can turn off the self-registration feature in our user pool, thus mandating that only administrators have the authority to create user profiles. Additionally, should we wish to restrict sign-ins through the user pool, it is possible to deselect our user pool as an IdP within our app clients.

We may also integrate Amazon Cognito with **Amazon Verified Permissions**, a refined authorization service designed for enforcing role and attribute-based access control in applications that utilize Amazon Cognito for authentication. Verified Permissions evaluates a user's attributes against the established access rules for a given resource based on their identity or access token. It then issues an authorization decision – either granting or denying access. This service allows for the centralization of authorization across all your applications and resources into one **policy store**. Policies are formulated in **Cedar**, an open source language specifically designed for crafting access control protocols.

We created a user pool from the console in this recipe. We can also create a user pool with the `aws cognito-idp create-user-pool` CLI command, passing `pool-name` and `region`. Even though we can directly specify all user pool settings from the command line by referring to the documentation, it is an easier and safer approach to use an input JSON file with all the configurations specified.

## See also

- We can read more about Amazon Cognito at <https://www.cloudericks.com/blog/getting-started-with-amazon-cognito>.
- We can read more about Amazon SNS service at <https://www.cloudericks.com/blog/getting-started-with-amazon-sns-service>.
- We can read more about Amazon Verified Permissions at <https://www.cloudericks.com/blog/getting-started-with-amazon-verified-permissions>.
- We can learn about integrating Amazon Cognito with Amazon Verified permissions at <https://www.cloudericks.com/blog/integrating-amazon-cognito-with-amazon-verified-permissions>.

## Using identity pools to access AWS resources

As discussed earlier in this chapter, Amazon Cognito serves two primary use cases. While user pools help us in handling identity and access management for our applications, identity pools extend this functionality to provide temporary AWS credentials, enabling secure access to various AWS services without the need for long-term keys. In this recipe, we will delve into the use of identity pools for effective access to AWS resources. Let's embark on a deep dive into the realm of identity pools and leverage their capability to weave identity pools into our applications, ensuring a secure and scalable method to access the AWS resources, making use of temporary AWS credentials.

## Getting ready

To complete this recipe, we need to ensure that the following additional requirements are in place in addition to those mentioned in the *Technical requirements* section:

- **An Amazon Cognito user pool:** An Amazon Cognito user pool serves as the identity backbone for our identity pool. We created one in the *Working with Amazon Cognito user pools* recipe earlier in this chapter.
- **App Client with ALLOW\_USER\_PASSWORD\_AUTH:** We need to set up an app client configured with the ALLOW\_USER\_PASSWORD\_AUTH flow. This can be accomplished by navigating to the user pool settings in the AWS Management Console, selecting the **App integration** tab, and creating an app client. Alternatively, we can include the authentication flow during the initial App Client setup as outlined in the *Working with Amazon Cognito user pools* recipe.
- **An Amazon S3 Bucket:** We need an S3 bucket with default options as discussed in the *Technical requirements* section of *Chapter 2* to demonstrate user pools. I will be using a bucket name of myawsbucket in the us-east-1 region. S3 bucket names must be globally unique; choose an available name and replace my bucket name with your chosen bucket name.

Next, we will see how we can implement Cognito identity pools.

## How to do it...

We can use identity pools for effective access to AWS resources as follows:

1. Navigate to the **Amazon Cognito** service in the AWS Console and search for the Cognito user pool that we created. Navigate to the **Users** tab and click on the **Create user** button.

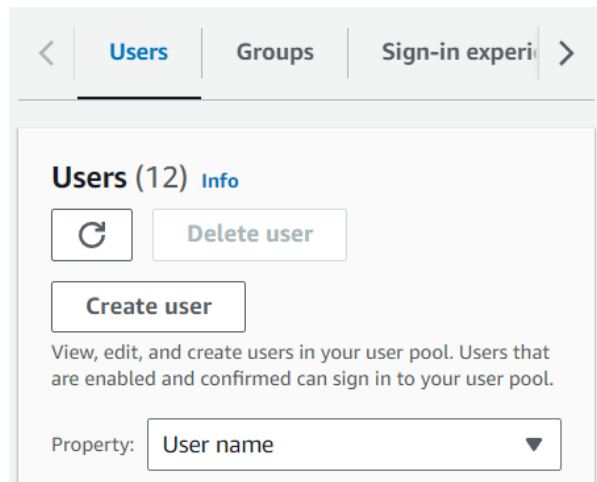


Figure 9.17 – The Create user option

2. Provide the user profile details: set **User name** to `testuser` and **Email address** to `testuser@cloudericks.com`. For **Temporary password**, select the **Set a password** option, enter a secure password, and finally, click on **Create user**.

Invitation message

Info

Configure invitation message templates in the [Messaging tab](#)

☒ Don't send an invitation

☐ Send an email invitation

User name

User name is an required attribute based on your user pool and above configurations.

testuser

Email address - optional

Enter this user's email address. A user's email address can be used for sign-in, account recovery, and account confirmation.

testuser@cloudericks.com

☐ Mark email address as verified

Temporary password

Amazon Cognito will send the password you generate to the user in an email message.

☒ Set a password

☐ Generate a password

Password

Enter a temporary password for this user. The temporary password will be sent to the user in their invitation message.

.....

☒ Show password

Cancel

Create user

Figure 9.18 – The details for creating a user

**Important note**

To prevent encountering MFA challenges when authenticating this user via the AWS CLI for an app client, we should choose the **No MFA** option in the **Configure security requirements** settings while creating the Cognito user pool.

3. In the AWS Management Console, go to the **Amazon Cognito** service.
4. Click on **Identity pools** and then **Create identity pool**.
5. Configure the identity pool by selecting **Authenticated access** and **Amazon Cognito user pool** under **Authenticated identity sources**. Click on **Next** to reach the **Configure permissions** page.

Configure your identity pool to generate credentials for users authenticated by third parties, and optionally, unauthenticated guests.

☒ **Authenticated access**

Issue credentials to authenticated users from trusted identity providers.

☐ **Guest access**

Issue guest-access credentials to anyone with internet access. Use guest access with AWS resources such as public APIs and graphics assets.

**Authenticated identity sources** [Info](#)

Configure identity providers to be the source of your authenticated identities. Amazon Cognito issues temporary credentials in exchange for tokens or assertions from your providers.

☒ **Amazon Cognito user pool**

Issue credentials to users who authenticate through an Amazon Cognito user pool. Your users can sign in to a user pool using the built-in user directory or through a third-party identity provider.

☐ **Facebook**

Exchange AWS credentials for Facebook OAuth tokens.

☐ **Google**

Exchange AWS credentials for Google OAuth tokens.

☐ **Apple**

Exchange AWS credentials for Apple OAuth tokens.

☐ **Amazon**

Exchange AWS credentials for Amazon OAuth tokens.

☐ **Twitter**

Exchange AWS credentials for Twitter OAuth tokens.

☐ **OpenID Connect (OIDC)**

Exchange AWS credentials for OAuth tokens from a custom OpenIDConnect (OIDC) provider.

☐ **SAML**

Exchange AWS credentials for assertions from a custom SAML provider.

☐ **Custom developer provider**

Issue credentials to users who authenticate with your own developer provider.

[Cancel](#)[Next](#)

Figure 9.19 – Configuring the authentication providers



- On the **Configure permissions** page, select **Create a new IAM role** and specify IAM roles for authenticated users by giving a role name. We can use MY\_IDP\_AUTHROLE as the **IAM role name**. Click on **Next**.

**Authenticated role** [Info](#)  
Configure the default IAM role that your users assume through your identity pool.

**IAM role**  
Choose a role in your account, or have Amazon Cognito create a new one for you.

☒ **Create a new IAM role**  
Create an IAM role with basic permissions and a trust relationship with your identity pool.

☐ **Use an existing IAM role**  
Choose a role in your account that you have configured to trust cognito-identity.amazonaws.com.

**IAM role name**  
Enter a name for your new IAM role.  
  
Role names may be up to 64 characters long and can use alphanumeric characters, as well as the following special characters: +,=,@-\_  

You're creating an IAM role with initial minimum permissions and a trust relationship with your identity pool. After you create your identity pool, add permissions in the IAM console.

**► View policy document**  
View the policy document that Amazon Cognito has generated.

Cancel

Previous

Next

Figure 9.20 – Selecting an authenticated role

- Enter the values for **User pool ID** and **App client ID**.

### User pool details [Info](#)

Select a user pool ID and app client ID.

User pool ID

App client ID

Figure 9.21 – The user pool details

Leave the other values as default and click on **Next**.

### Attributes for access control [Info](#)

Map claims from your identity provider to principal tags that control access to your AWS resources. Amazon Cognito applies principal tags to your user's temporary session.

**i** Amazon Cognito can pass attributes for access control session tags to an IAM role only when you permit the action `sts:TagSession` in the role trust policy.

**i** Attributes for access control settings for Amazon Cognito user pools apply to all app clients that you link from the same user pool. You can apply different user pool attributes for access control settings only to separate user pools.

Claim mapping

- ☒ Inactive
- ☐ Use default mappings
- ☐ Use custom mappings

Cancel

Previous

Next

Figure 9.22 – The attributes for access control

8. Give a name for the identity pool and click on **Next** to finish the configuration.

The screenshot shows the 'Create identity pool' configuration page in the AWS IAM console. At the top, there is a 'Name' field with the text 'MYIDENTITYPOOL' entered. Below the field, a note states: 'Identity pool names are limited to 128 characters or less. Names may only contain alphanumeric characters, spaces, and the following special characters: + = , . @ -'. The next section is 'Basic (classic) authentication' with an 'Info' link. It contains a checkbox labeled 'Activate basic flow' which is currently unchecked. The following section is 'Tags (0) - optional' with an 'Info' link. It states 'Tags are key-value pairs you can add to AWS resources to help identify, organize, or search for resources. Choose any tags you want to associate with this identity pool.' Below this, it says 'No tags associated with the resource.' and there is a button labeled 'Add new tag'. A note below the button says 'You can add up to 50 tags.' At the bottom right of the form, there are three buttons: 'Cancel', 'Previous', and 'Next' (which is highlighted in orange).

Figure 9.23 – The final configuration page

9. On the **Review and create** page, review everything and click on **Create identity pool**.
10. We should see a prompt stating that the role has been created successfully; click on **View role**.

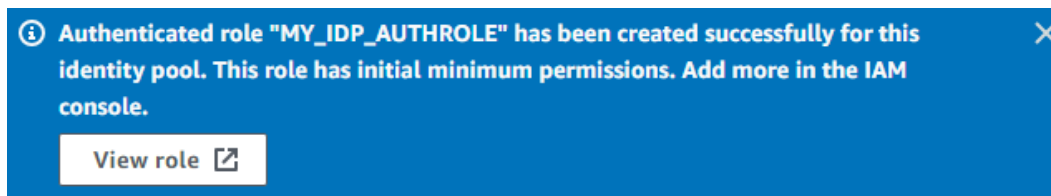


Figure 9.24 – The prompt after successful role creation for the identity pool

11. The role for authenticated users should have permission to access S3 buckets. Make sure to grant access rights for the **AmazonS3ReadOnlyAccess** action. Click on the **Add permissions** dropdown and select **Add policy**. Search for the **AmazonS3ReadOnlyAccess** policy, select it, and click on **Add permissions**.

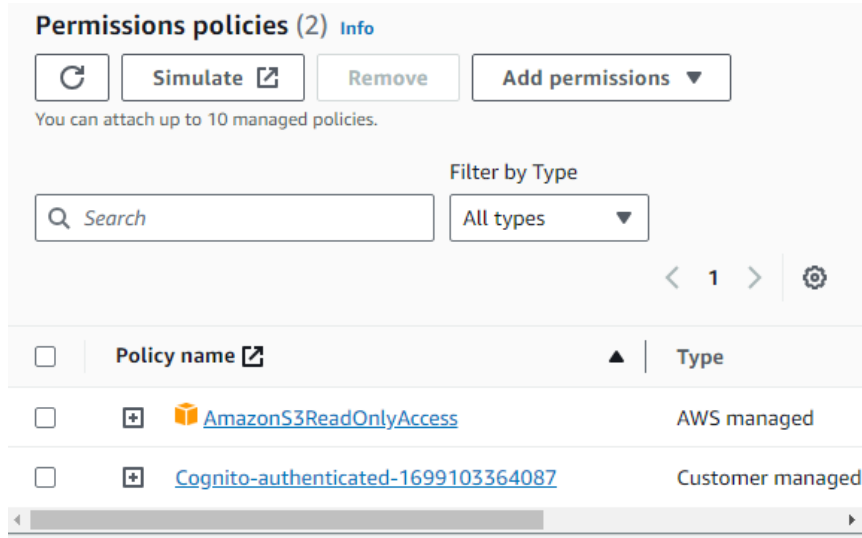


Figure 9.25 – The permission policies

12. To authenticate a user in the user pool, we can use the **Amazon Cognito Identity SDK** or directly use AWS CLI's `cognito-idp` commands. For this, go to AWS CloudShell and run the following command by providing the `<username>`, `<password>`, `<client-id>`, and `<region>` of the user we want to authenticate:

```
aws cognito-idp initiate-auth
--auth-flow USER_PASSWORD_AUTH
--auth-parameters USERNAME=<username>,PASSWORD=<password>
--client-id <client-id> --region <region>
```

We should get a challenge response asking us to give a new password.

```
{
  "ChallengeName": "NEW_PASSWORD_REQUIRED",
  "Session": "AYABeOnAspBB0LG731L_5-TOMhEAHQABAAdTZXJ
RiZQC4AQIBAHgDHnKSW2nDRJSDSLf55TGFyX5On_wV32whMfiMxuCE
SGHZYYcRPGAEBLi7QwNAhbi11nwilV4uaSNrOxhYs9RceTZz6FXJG_Q
ysjfgiW631YLiMq1dQFezfs1_Fjtk2kpZe76p2Ka3ylzNi9uU0y4uCV
LGRSVeghQ",
  "ChallengeParameters": {
    "USER_ID_FOR_SRP": "michael",
    "requiredAttributes": "[]"
  }
}
```

Figure 9.26 – The challenge response asking us to give a new password

13. We need to respond to the challenge to give the user a new password using the following command by giving the `user-pool-id`, `client-id`, `NEW_PASSWORD`, `USERNAME`, `session`, and `region`. The session should be the one we received in response to the previous command:

```
aws cognito-idp admin-respond-to-auth-challenge
--user-pool-id <user-pool-id> --client-id <client-id>
--challenge-name NEW_PASSWORD_REQUIRED --challenge-responses
NEW_PASSWORD=<new-password>,USERNAME=<username>
--session <session> --region <region>
```

We should get a screen showing the **RefreshToken** and **IdToken**. We should get the screen as follows; take note of the **IdToken** value.

```
YyIsInRva2VuX3VzZSI6ImFjY2VzcyIsInNjb3B1IjoieYXZlLnVzZ25pdG8uc2lnbmLuLnVzZXIuYWRtaW4iLCJhdXRoX3RpbWUiO
ZSI6Im1pY2hhZWwifQ.p33J7oOoaQ7grB_i-Hwcs0MTKD7sR8pU1IBtAhwaYyv_x-PwWqapB6GB0pAQdIeSmvQnnxU8NFVHagnCG
P5a68L476eSG-BelM87j14DHIXT_DFzgyRthbNrKPI6lswUnF_x7e54_monW0s5qSQq8zGvlgpDfhv9GXCNvShk9I44Fuocyfy7r
"ExpiresIn": 3600,
"TokenType": "Bearer",
"RefreshToken": "eyJjdHkiOiJKV1QlCjlmMi0iJBMjU2R0NNIiwiaWxnIjoiaUINBLU9BRVAifQ.QB7FPuvMyOeI7
Ih82grKCB9oG0VaDjqpnNg_iPTOna726QUZG8yGMD0jUub3n_racqEKYHwpJzu59y2T_vjX2M1C-nCn7kC8r22HfyPoJ8ZIZB3Gj
oSFNB5T8YEK7ATP8MMteCz9BSH-eFZjvv25zQ0I-nZx4ccdnITmRKembXkligQBFHi05PqkoRtk2-yn5L2dC_fzuSILo1p2ixj7S
YZRMnQG5WN9TGJdaBbW5wsMhs5RCfn88GpmivklzcmcgXTZPLAa21sSpDjrrHqRJRjg5_8JUId6BMyJUEwUSxxNUG2e04fuiEdKrB
Xq9ompQ7pa86V2hA-Q12a7QfoRKeRa40BCDsJqHm16Tmz2w_9KZkHNPuJhJ7DiyNXc5twOHwtJiD1aJs6Tf7_a8tc0gTaIafBnUj9
K7d4Q9iub1TJW4w4ooVBu4ggydtzn589QuDGx-rve74B9egkd2d_vu-Fv1rx-ocbttr3FMZ9WaiRmxm_yzoPpHbZw6V008xBoAfJ
eG5JKAFCpb1ILQZrwXF1n1A3B0eddKasSKn14nW2IXW51SXYJWiaDZfBR8hh74Mg8-5SrGcuJ7yzMXFuQXsWIrW9tpvc355rVGd1i
Ttetdw5FWQsd-zxG7y7kbjsk9IVieoZwh5A1dXKJ1MtbMakdZdkeKietqozAZ63Mf6C4cpzG26QyQuqf9AA1d0BANLpt6zkGo02Px
"IdToken": "eyJraWQiOiJlaktCcTEwQmg5N1pkbDdPa01keUxyc1o5bEV1a3RYQ1crT0RteFRpYV1zPSIsImFsZyI6I
bmF3cy5jb21cL3VzLWVhc3QtMV9VZUwzSFB0U3QilCjlb2duaXRvOnVzZXJvYm1lIjoibWljajGF1bCIzIm9yaWdpb19qdGkiOiIzY
M2MwLTkxZDEtNTA2YzE4NmRhZThjIiwiwidG9rZW5fdXN1IjoiaWQiLCJhdXRoX3RpbWUiOjE2OTkwOTc4MTksImV4cCI6MTY5OTETwM
bSj9.q9I3Z69pXsWkIP71YDDUgZsaHduAPVe_AeITaZdUYAh-JwFIgORIIY_BKIy5GX7Ioa_z-37N_AS0w5Qoq2duzp8hHkDouTwx
JzoX6ZB8mG57uBxAiu1v4j9ws9Zeggy5lhccfwN1aCIz0Rf1GPFxasD8Io0rXmVGu9ViAzB218K1tKBZOU_3vYsyBu2gWoYV9_GaM
}
```

Figure 9.27 – The response after the password change

14. To get **IdentityId**, run the following command by providing the `<identity-pool-id>`, `<region>`, and `<user-pool-id>` of the identity pool that we created previously, as well as the `<id-token>` we received from the previous step:

```
aws cognito-identity get-id
--identity-pool-id <identity-pool-id>
--logins "cognito-idp.<region>.amazonaws.com/<user-pool-id>=<id-
token>" --region <region>
```

```
Users\mike>aws cognito-identity get-id --identity-pool-id us-east-1:a24f6efc  
pkBdDpa01keUxyc1o5bEV1a3RYQ1crT0RteFRpYVlzPSiIsImFsZyI6IlJTMjU2In0.eyJzdWIIOiO.  
cl3VzLWWhc3QtMV9vZWUwzSFB0U3QiLCJjb2duaXRvOnVzZXJyYW1lIjoibWljagF1bCIsIm9yaWw  
IjoimjI5ZTVmYzctOWM3Ny00NDM2LWJlcmMtNjk3ZjnlMjA3MjdhdhIiwidG9rZW5fdXNlIjoiaWQ.  
GizYmUmNmVlZDEiLCJlbWpfbpCI6Im1pY2hhZWxfaxRlZ2JlQHlaG9vLmNmNbSj9.KWAQJLE4HLg  
SwusbfTWtiSmXYNZ7ThyRfwZb_G_FOkQcgb_mJlqQpsYmjS89Z7KzaDb02L78GjRf2VGULGH5ixf  
guzg1G6QwKyQ" --region us-east-1
```

"IdentityId": "us-east-1:42b3aeb5-56a7-4d89-9479-7fc2f83eaa0e"

15. Use the **IdentityId** obtained from the previous step to get temporary AWS credentials:

is command will return temporary AWS access and secret keys, as well as a session token.

```
{
  "IdentityId": "us-east-1:42b3aeb5-56a7-4d89-9479-7fc2f83eaa0e",
  "Credentials": {
    "AccessKeyId": "ASIA6H600BX3GCGKFUGC",
    "SecretKey": "fEI70x8Mzq2cS3piGYRzU1kwVJ5tNMPVTY1PUM8D",
    "SessionToken": "IQoJb3JpZ2luX2VjEQCAQXVzLWVhc3QtMSJGMEQCIBFXAMe+ATXKqEEq/ddAzTyWZUVmN6xjtSoxLM3hXyq6dlEUI2n1UCa+zdYUZFuyLEeEMo/NFkukVAHddmiNZSATDZicvQInvfafpTvRNF3U6bAakxsrgCqSht7Tn6G2anUEDaQ/Z7PfigGSuEoUue:1qymW3Mj7dhNENjCAofbCkr620EfTR1BS6ypkeL6NFA44MH4Ue0soeuSfvfLxQl1f6xomPmIcZnJedEyADMEZp5CVDFdlaSz2NKNL v/FNRO5AHYDvBqHBv7aHL26TKfCMnQMB2COfTUH014Z4AHLMvn4owldTU9163SmiaE4DGGW3S08+Vn9N2wm9AfgabV5i/sXcEWGFz9/nE90/YcdwF4Cvok0avdJoI5dyBdlTUancNs4BdoneGS5IABLPgluHx29Bf+BU4jIZQp14NZ7wjAQuj4ssWJHj2"
    "Expiration": "2023-11-04T13:52:58+01:00"
  }
}
```

Figure 9.29 – The response to the `get-credentials-for-identity` command

16. Configure the AWS CLI with the temporary credentials obtained in the previous step.

```
aws configure set aws_access_key_id <access-key>
aws configure set aws_secret_access_key <secret-key>
aws configure set aws_session_token <session-token>
```

These commands do not return any response as shown below:

```
C:\Users\mike>aws configure set aws_access_key_id "ASIA6H600BX3GCGKFUGC"
C:\Users\mike>
C:\Users\mike>aws configure set aws_secret_access_key "fEI70xBMzq2cSjpiGYRzUlkwVJ5tNMPVTY1PUM8D"
C:\Users\mike>
C:\Users\mike>aws configure set aws_session_token "IQoJb3JpZ2luX2VjECQaCXVzLWVhc3QtMSJGMEQCIBFXAM
E0DYwNjk2NiIMoI7bWwF0i0es+ATXKqEEq/ddAzTywZUVmN6xjtSoxLM3hXyq6dlEUI2n1UcA+zdYUZfuyLEeMo/NFkukVA
6oxoGzQ0zy1KJcxz0gn40c2EgJddmiNZSATDZicvQInvfafpTvrNF3U6bAakxsrGcQsht7Tn6G2anUEDaQ/Z7PfigG5uEoUue
SSgljopZBxPuryLL12uDGGA2YN1qymW3Mj7dhNENJKAofbCkr620eFTR1BS6ypkeL6Nfa44MH4Ue0soeuSfvFLxQl1f6xomPmI
S9qPrWDHw5hvUw0e0cnE+1uamSZnJedEyADMEZp5CVFD1aSzNKNLlV/FNROSaBYDvBqHBv7aHL26IKFcMnQmB2COFTUH014Z4H
XQOpJ0u/eJ4NmEuJh0v6QhmyOeLMvn4oWldTU9163SmiaE4DGGW3S08+Vn9N2wm9AfgabV5i/sXcEWGFz9/nE90/YcdwF4Cvo
ATELucn7Cyzb3m1hFPy11h2G+b0avdJoISdyBdl1tUAncNs4BdoneGS5IABLPGluHx29Bf+BU4jIZQp14NZ7wjAQuj4sswJHj2
```

Figure 9.30 – Configuring the AWS CLI

17. We can now use the AWS CLI to list our S3 buckets using the `aws s3 ls` command.

```
C:\Users\mike>aws s3 ls
2023-11-02 17:55:03 amplify-petstoresample-devn-165457-deployment
2023-11-03 10:13:31 amplify-petstoresample-devo-91326-deployment
2023-11-03 10:14:14 amplify-petstoresample-marla-91410-deployment
2023-10-16 10:14:04 myawwsbucket
2023-10-25 16:04:14 mybucketr635363
```

Figure 9.31 – The response to the `s3 ls` command

18. Navigate back to our identity pool and click on **User statistics**. We should see that we have been authenticated successfully via the identity pool.

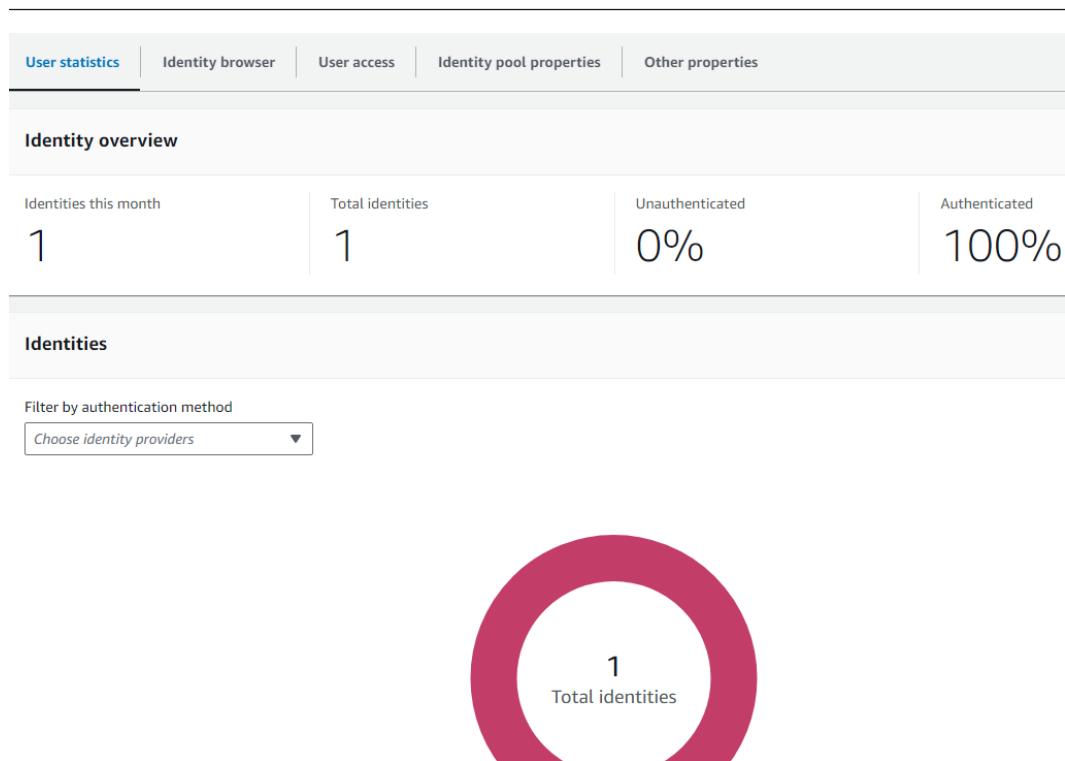


Figure 9.32 – The user statistics

In this recipe, we made use of an identity pool to access AWS resources.

## How it works...

To access Amazon S3 using an identity pool with Amazon Cognito user pool as the IdP and list S3 buckets via the AWS CLI, we first created a user pool and a user within it. Then, we set up an identity pool, defined IAM roles, and configured permissions for authenticated users, including listing S3 buckets.

Next, we obtained an ID token through the AWS CLI's `cognito-idp` commands, initiated authentication, and got an ID from the identity pool. With the obtained ID, we retrieved temporary AWS credentials. After configuring the AWS CLI with these credentials, we used it to list our S3 buckets.

This process allows secure and controlled access to S3 resources via identity-based authentication, ensuring that users are authorized to perform specific actions on the S3 service.



## There's more...

Let us see some more concepts related to Amazon Cognito that will help us further understand the service:

- **Social IdPs:** Both Cognito user pools and identity pools can be configured to allow users to sign in using social IdPs such as Facebook, Google, or Amazon. This extends the authentication options for our application and provides a seamless sign-in experience for users who may prefer to use their existing social media accounts.
- **MFA:** We can enable MFA in both Cognito user pools and identity pools to enhance security. With MFA, users are required to provide an additional authentication factor, such as a one-time code from a mobile app or SMS, in addition to their password. This significantly strengthens the security of user authentication.
- **Custom authentication challenges:** Cognito user pools allow us to set up custom authentication challenges. This can be useful for scenarios where we need users to complete specific tasks or provide additional information during the authentication process. We can customize the challenge flow to match our application's requirements.

Exploring every feature of Amazon Cognito in depth would need a book of its own. Therefore, additional links are provided in the *See also* sections of both this recipe and the previous one to allow you to explore further.

## See also

- We can learn about using Federated IdPs with Amazon Cognito at <https://www.cloudericks.com/blog/using-federated-identity-providers-with-amazon-cognito>.
- Read more about troubleshooting security incidents using AWS Cognito logs at <https://www.cloudericks.com/blog/troubleshooting-security-incidents-using-aws-cognito-logs>.

## Using AWS Simple AD for creating a lightweight directory solution

AWS Simple AD, powered by a Samba 4 Active Directory Compatible Server, is a lightweight standalone managed directory service provided by AWS, catering especially to small and medium-sized businesses seeking the functionality of directory services without the complexity of managing on-premise infrastructure. Available in two sizes, the **Small** option is intended for environments with up to 500 users (or approximately 2,000 objects, including users, groups, and computers), while the **Large** option accommodates environments with up to 5,000 users (or roughly 20,000 objects).

Simple AD provides essential services such as user account management, group membership organization, group policy application, secure Amazon EC2 instance connections, and Kerberos-based **Single Sign-On (SSO)** functionality. It is compatible with various applications and tools dependent on Microsoft Active Directory, facilitating access to AWS applications such as WorkSpaces, Amazon WorkDocs, and Amazon WorkMail for users. However, Simple AD does not support certain services, including Amazon AppStream 2.0, Amazon Chime, or Amazon RDS for SQL Server and Oracle.

## Getting ready

We need the following to successfully complete this recipe:

- A working AWS account and a user, as described in the *Technical requirements* section.
- A VPC configured with at least two subnets across different availability zones following the *Setting up VPC plus VPC resources with minimal effort* recipe from Chapter 5.

## How to do it...

We will use the following steps:

1. Log into the AWS management console and go to the **Directory Service**. We should see a screen like the following.

### Set up directory

Choose which directory best fits your business needs and we'll walk you through how to set it up.

AWS Managed Microsoft AD ▼

**Set up directory**


Not sure which type of directory you need  
? We can help you choose. [Learn more](#) 

Figure 9.33 – The Set up directory screen

2. Select **Simple AD** from the drop-down menu, as shown in *Figure 9.34*, and then click on the **Set up directory** button shown in *Figure 9.33*. Click on **Next**.

### Set up directory

Choose which directory best fits your business needs and we'll walk you through how to set it up.

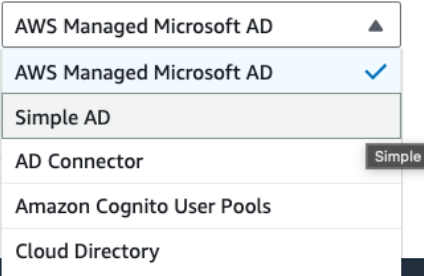


Figure 9.34 – The Set up directory drop-down menu

3. On the **Select directory type** page, make sure that **Simple AD** is selected and click **Next**.
4. On the **Enter directory information** page, under **Directory size**, choose **Small**. For **Directory DNS name**, enter `corp.cloudericks.com`, and for **Directory NetBIOS name - optional**, enter `CORP`.

#### Directory type

Simple AD

#### Directory size [Info](#)

Simple AD is available in the following two sizes:

☐ Small

~USD 36.0000/mo (USD 0.0500/hr)

Small directories can have up to 2000 objects, including ~500 users, groups and computers each.

☐ Large

~USD 108.0000/mo (USD 0.1500/hr)

Large directories can have up to 20,000 objects, including ~5000 users, groups and computers each.

#### Directory DNS name

A fully qualified domain name. This name will resolve inside your VPC only. It does not need to be publicly resolvable.

`corp.cloudericks.com`

#### Directory NetBIOS name - *optional*

A short identifier for your domain. If you do not specify a NetBIOS name, it will default to the first part of your Directory DNS name.

`CORP`

Maximum of 15 characters, can't contain spaces or the following characters: ``\/:*?"<>|``. It must not start with `..`

Figure 9.35 – Configuring Simple AD

5. Scroll down and provide a value for **Administrator password** and confirm the password in the **Confirm password** field. For **Directory description - optional**, provide the `Cloudericks corporate directory` description for the directory, and click **Next**.
6. For **VPC**, select a VPC with at least two subnets, as discussed in the *Getting ready* section. Select two of its public subnets under **Subnets** and click **Next**.
7. Under **Subnets**, choose the subnets for the domain controllers. Ensure that they are in different Availability Zones and click on **Next**.
8. On the **Review & create** page, review all details and click on **Create directory**. The status of our directory will be **Creating** and once creation is complete, it will change to **Active**.

## How it works...

AWS Simple AD provides a managed Active Directory-compatible directory service, offering users a straightforward solution for user authentication, group management, and domain join operations in the AWS cloud. Users can easily create a Simple AD directory through the AWS Management Console or CLI, specifying domain details and size requirements.

Once created, Simple AD supports standard directory features such as LDAP authentication and Kerberos-based SSO, allowing organizations to manage user accounts and access controls efficiently. This service is particularly beneficial for small and mid-sized businesses seeking a cost-effective and hassle-free Active Directory solution without the complexity of self-managed infrastructure.

AWS Simple AD allows efficient user and group management, supports Kerberos-based SSO for seamless access to multiple applications, and integrates with various AWS services such as Amazon EC2, RDS, and WorkSpaces. Simple AD also supports LDAP, making it suitable for applications requiring directory services for authentication and information retrieval.

## There's more...

Here are the different ways we can use AWS Simple AD:

- **User and group management:** Simple AD allows you to manage users and groups efficiently. You can create, modify, and delete user accounts, as well as organize them into groups for easier access control and permissions management.
- **SSO:** Simple AD supports Kerberos-based SSO, enabling users to log in once and gain access to multiple applications and services without re-entering their credentials.
- **Lightweight Directory Access Protocol (LDAP):** Simple AD supports LDAP, allowing applications that require directory services to authenticate and retrieve information from the directory.
- **Integration with AWS services:** Simple AD integrates seamlessly with various AWS services, enabling you to manage access and permissions for resources within your AWS environment.

- **Amazon EC2:** Manage access to EC2 instances using domain credentials.
- **Amazon RDS:** Integrate with RDS instances for database authentication.
- **Amazon WorkSpaces:** Use Simple AD to manage user access to virtual desktops.
- **Cost-effective directory services:** Simple AD provides a cost-effective solution for small to medium-sized businesses that need basic directory services without the need for a full Active Directory setup.

In this recipe, we explored Simple AD, a cost-effective and straightforward directory service within AWS ideal for small to medium-sized businesses. AWS also supports other directory solutions, including **AWS Managed Microsoft AD**, which offers fully managed Active Directory services; **AD Connector**, a gateway to redirect directory requests to on-premises AD; **Amazon Cognito User Pools**, providing user sign-up and sign-in for web and mobile apps; and **Amazon Cloud Directory**, a scalable directory for managing hierarchical data relationships.

## See also

- Read more about Simple AD at <https://www.cloudericks.com/blog/getting-started-with-aws-simple-ad>.
- Read about different directory solutions in AWS such as AWS Managed Microsoft AD, Simple AD, AD Connector, Amazon Cognito User Pools, and Amazon Cloud Directory at <https://www.cloudericks.com/blog/exploring-aws-directory-solutions>.

## Using Microsoft Entra ID as the identity provider within AWS

In today's tech landscape, numerous organizations are adopting a multi-cloud strategy, utilizing different public clouds such as AWS and Azure for deploying applications. A common practice among these organizations is managing user identities with Microsoft Entra ID. The AWS IAM Identity Center offers a straightforward way to link our AWS accounts with Microsoft Entra ID. This integration yields two significant benefits: centralized management of identities and enhanced user experience.

Centralizing identity management in one location simplifies the responsibilities of IT teams and strengthens security protocols. Additionally, it relieves users of the hassle of handling multiple login credentials, streamlining the sign-in process and reducing the demand for IT support. In this recipe, we will delve into integrating Microsoft Entra ID with AWS IAM Identity Center, using Microsoft Entra ID as the IdP. We will conclude by demonstrating this setup using an Entra ID user to manage an S3 bucket, as an AWS resource.

## Getting ready

To complete this recipe, we need to ensure that the following additional requirements are in place in addition to those mentioned in the *Technical requirements* section:

- It's essential to have an AWS account with AWS IAM Identity Center set up. For guidance on setting up IAM Identity Center, refer to the *Chapter 1* recipes.
- An active Microsoft Entra ID tenant is also a must-have.
- An administrator user in the Microsoft Entra ID tenant is essential for performing the configuration steps within this recipe.
- A demo user with data for the attributes first name, last name, display name, and user principal name is also needed to test the configuration. Let us call the demo user `demouser1`.

### Important note

To successfully integrate users into the AWS IAM Identity Center through the AWS SSO Enterprise application in Azure, the users need values for the following attributes: first name, last name, display name, and user principal name. Without these attributes, the provisioning process may encounter errors, hindering the smooth and secure transition of users into the AWS environment.

Next, we will see how to use Microsoft Entra ID as the IdP within AWS.

## How to do it...

We will first download the metadata file from AWS, configure Microsoft Azure for IAM Identity Center integration with that metadata file, and continue with AWS IAM Identity Center set up in AWS. After that, we will go back to Azure to complete the configuration and finally will verify users within AWS.

### Downloading the metadata file from AWS

1. Log in to the Management Console of an AWS account with permission to manage IAM Identity Center and go to the IAM Identity Center service.
2. Click on **Settings** in the left sidebar.
3. On the **Settings** page, locate the **Identity source** section, click on the **Actions** pull-down menu, and select **Change identity source**.
4. On the **Change identity source** page, choose **External identity provider** and click on **Next**.


**Important note**

On the **Change identity source** page, there are currently three options: **Identity Center directory**, **Active Directory**, and **External identity provider**. If we select the **Identity Center directory** option, it will allow for the management of users and groups within IAM Identity Center, with users logging in through the AWS access portal. The **Active Directory** option is used for managing users and groups within AWS Managed Microsoft AD. We also have the flexibility to connect IAM Identity Center to an existing Active Directory using either AWS Managed Microsoft AD or AD Connector. In this scenario, users also sign in through the AWS access portal. The third option, **External identity provider**, is tailored for managing users and groups through an external IdP such as Microsoft Entra ID. Here, users first sign in to the IdP's sign-in page, as we will see in this recipe, and are then redirected to the AWS access portal, where they can access their AWS accounts and cloud applications.

Once we click **Next**, we should see the **Configure external identity provider** page, which looks similar to the following:


## Configure external identity provider

### Service provider metadata


 **Download metadata file**

Your identity provider (IdP) requires the following IAM Identity Center certificate and metadata information to trust IAM Identity Center as a service provider. You can copy and paste this information, type it in the service provider configuration interface for your IdP, or download the IAM Identity Center metadata file and upload it to your IdP.

AWS access portal sign-in URL

 <https://awsseccb.awsapps.com/start>

IAM Identity Center Assertion Consumer Service (ACS) URL

 <https://us-east-1.signin.aws.amazon.com/platform/saml/acs/836c1a1f-4754-412d-a723-29bcdbdc9640>

IAM Identity Center issuer URL


 <https://us-east-1.signin.aws.amazon.com/platform/saml/d-90679fa661>

Figure 9.36 – The Configure external identity provider page

5. On the **Configure external identity provider** page, within the **Service provider metadata** section, click on **Download metadata file** and save it on the computer. This file will be used when configuring the external IdP (e.g., Microsoft Entra ID).

The metadata file we downloaded from IAM Identity Center includes the IAM Identity Center certificate and metadata details necessary for our IdP, which in this recipe is Microsoft Entra ID, to establish trust with IAM Identity Center as a service provider. Alternatively, instead of downloading, we can simply copy this information directly from the webpage and manually enter it into the service provider configuration section of our IdP.

### ***Configuring Microsoft Azure for IAM Identity Center integration***

We can configure Microsoft Azure for IAM Identity Center integration as follows:

1. Sign in to the Microsoft Azure portal as a user with the **Application Administrator** role or a superset role.
2. Navigate to **Microsoft Entra ID** and click on **Enterprise applications** from the left sidebar.
3. Click **New application**, and on the **Browse Microsoft Entra Gallery** page, search for **AWS IAM Identity Center**.
4. Enter an application name and click on **Create**.
5. Open the application after it has been successfully created.
6. Click on **Single sign-on** from the left sidebar and select **SAML** as the SSO method.
7. Click on **Upload metadata file**, and in the **Upload metadata file** pane, click on the folder icon to select the metadata file that we downloaded in *Step 5* of the previous section. Then click on **Add**.
8. Click **Save** on the **Basic SAML Configuration** pane.
9. Scroll down to the **SAML Certificates** section of the page and download the **Federation Metadata XML** file.

We can now go back to the AWS Management Console on the **Configure external identity provider** page and continue with the steps required. References to detailed steps, with screenshots for the steps to be done on the Azure side, are given within the *See also* section of this recipe.



## Continuing with the AWS IAM Identity Center setup

We can continue the steps required on AWS IAM Identity Center from the AWS Management Console:

1. Navigate back to the **Configure external identity provider** page as we did in the *Downloading the metadata file from AWS* section.
2. In the **Identity provider metadata** section, click on **Choose file** under **IdP SAML metadata** and upload the file we downloaded in the *Configuring Microsoft Azure for IAM Identity Center integration* section.

### Identity provider metadata

AWS requires specific metadata provided by your identity provider (IdP) to establish trust. You may copy and paste from your IdP, type the metadata in manually, or upload a metadata exchange file that you download from your IdP.

#### IdP SAML metadata



✓ 2024-0-10\_10-047\_d-90679fa661\_sp\_saml\_metadata.xml


Size: 679 bytes

Last modified: Jan 10, 2024

Figure 9.37 – Uploading the identity provider metadata

3. Scroll down and click **Next**.
4. Scroll down, and within the **Review and confirm** section, type **ACCEPT**, as shown in the following figure, and click **Change identity source**.

## Review and confirm

 Review the following consequences of your requested identity source change:

- You are changing your identity source to use an external identity provider (IdP).
- IAM Identity Center will delete your current multi-factor authentication (MFA) configuration.
- All current permission sets and SAML 2.0 application configurations will be retained.
- IAM Identity Center preserves your current users and groups, and their assignments. However, only users who have usernames that match the usernames in your identity provider (IdP) can authenticate.
- You must complete your identity provider (IdP) SAML configuration for IAM Identity Center so that your users can sign in. Identity Center will use your IdP for all authentications.
- You must manage your multi-factor authentication (MFA) configuration and policies in your identity provider (IdP).
- You must add (provision) all users in your identity provider (IdP) who will use IAM Identity Center before they can sign in. If you enable System for Cross-domain Identity Management (SCIM) to provision users and groups (recommended), your IdP will be the authoritative source of users and groups, and you must add and modify them in your IdP. Without SCIM, you can provision users and manage groups in IAM Identity Center only; all provisioned usernames must match the corresponding usernames in your IdP.
- IAM Identity Center will keep your current configuration of attributes for access control. We recommend that you review your configuration and update it after you complete the identity source change.

Confirm that you want to change your identity source by entering **ACCEPT** in the field below.

[Cancel](#)[Previous](#)[Change identity source](#)

Figure 9.38 – Reviewing and confirming the identity source change

5. Within the settings page, we should now see a message for enabling **Automatic provisioning** with an **Enable** button. Click on **Enable**.

- Once the provisioning is enabled, we should see the **SCIM endpoint** and **Access token** values for our identity source, as shown in the following figure. Save this information.

**Download or copy the access token as this is the only time it will be shown**

You cannot recover it later. However, you can generate new tokens at any time. [Learn more](#)

SCIM endpoint

 <https://scim.us-east-1.amazonaws.com/f3vc5e7d636-e398-4b48-bacd-84ce3bd48976/scim/v2/>

Access token

 47133ace-68e3-4752-8342-a9892d965329:8764ae22-ff2c-401b-9c49-7c8242cbf3fa:HqLearRp3jtJ5R4rMqZ0hB92xwPtdM4ka6PGd2vK/yi0H+m2rs+ZxZZd7P66ukswiWvUSkUXsq7JhKqXpXLE9J1OgoRVDRC60JYpVnF5DqJcdB/8cs0qBb4Xzw7bXkDqSP3qfs/aQ6y/Xpqu8Op30ZSp3besLMecBY=:crvDbtZFG4NGcLlv26ZIZSfgctGwt/wKImNigXMKssA/DrLdi3e6HxUUMMVpfS7nN8O3JR+Ow+iWZbsGT00OaUQw4JCdrTMjoNnrpmYwwdoAydH531qfq7j1RbYRQ213Dpl80mTzOf2PKOtyvtMGxc7AL0CbziV8Xk795TsEjur+Q1zSyiRv8GeZLUIR3rfEUIVjGfl6nLyS8fwgjMHhZkUS6xYqlvOjgj7ANcYJluWKApcnbl9kVc7MI25Ekr dO8BxAfSG9Yjw7Obi9jxUWuxaF/31MD0MAIb4/JAg6vZhBJw/0IMgqYFzn6S+8biGB0paNKvcZUg/3EdKEg51+7A==

Hide token

Figure 9.39 – The access token

Let us now go back to the Azure portal and continue the configuration.

### ***Continuing with the configuration in Microsoft Azure***

Let us now go to the AWS IAM Identity Enterprise application we created in the *Configuring Microsoft Azure for IAM Identity Center integration* section and continue setting up:

- Under the **Manage** section on the left sidebar, click on **Provisioning**, and set **Provisioning Mode** to **Automatic**.
- Under the **Admin Credentials** section, enter the AWS Identity Center SCIM endpoint that we saved in *Step 6* from the previous section under **Tenant URL** and **Access token** under **Secret Token**.
- Click on **Test Connection**, and if this is successful, click **Save** under **Provisioning**.
- On the **Provisioning** tab, click on **Start provisioning**.
- Under the **Manage** section on the left sidebar, click on **Users and groups**, and then on the panel on the right side, click on **Add user/group**.
- Select the `demouser1` user that we created for the demo and click on **Assign**.

Once the provisioning is successful, we can head back to AWS IAM Identity Center to verify the user provisioning. It could take some time, usually up to around 30 minutes, for the users to appear in AWS. References to detailed steps with screenshots for the steps to be done on the Azure side are given within the *See also* section of this recipe.

Verifying Users on AWS IAM Identity Center

We can verify the automatic user provisioning in AWS IAM Identity Center as follows:

- 1. Navigate back to the IAM Identity Center dashboard.
- 2. Click on **Users** from the left sidebar. We should be able to see the demouser1 user in our AWS Identity Center. We can now log in to the AWS Identity Center using these user's Microsoft Entra ID credentials.

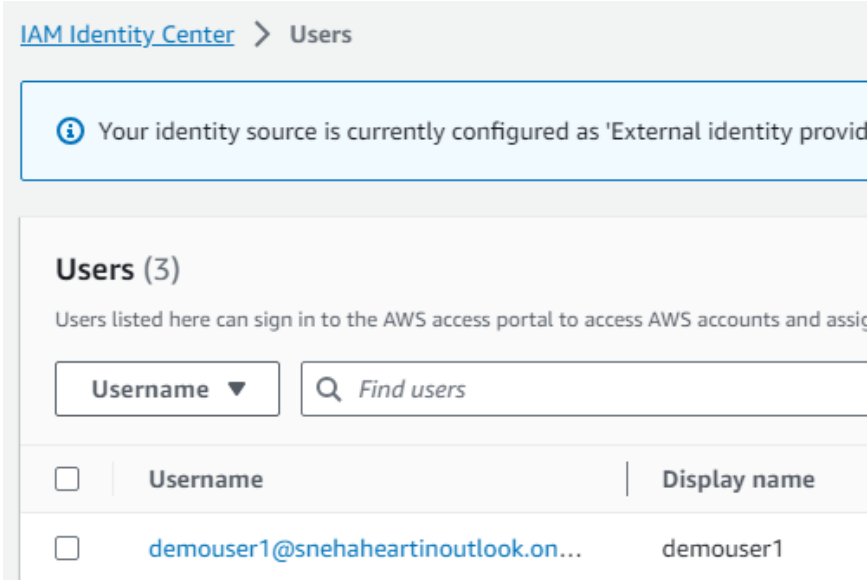


Figure 9.40 – Azure user in AWS

- 3. Click on **Settings** and copy the AWS access portal URL.
- 4. Paste this URL in another browser. We should be given a prompt to log in with our Microsoft Entra ID credentials. Enter the user's principal name and password.

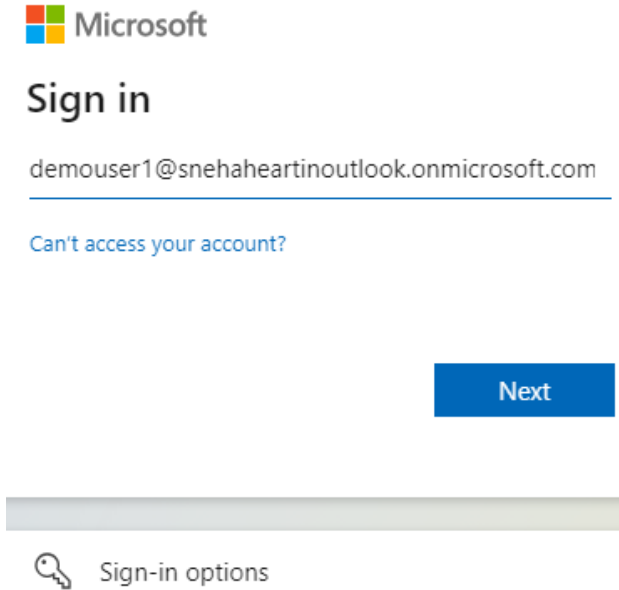


Figure 9.41 – Signing in as a Microsoft user

After entering the user's principal name and password, we should be redirected to the AWS account as a logged-in user.

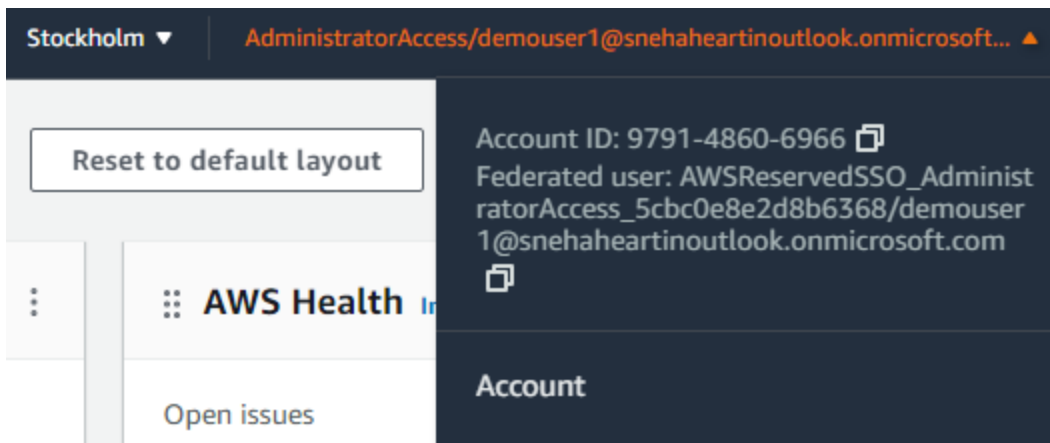


Figure 9.42 – The credentials of an Azure user in AWS account

We have used Microsoft Entra ID as our IdP in this recipe and logged in to an AWS account using the Microsoft Entra ID credentials. We will now need to provide the necessary permissions and access to this user using **permission sets**. We can also follow a similar approach that we did in this recipe to use other supported IdPs instead of Microsoft Entra ID.

## How it works...

SSO between AWS Identity Center and Azure Entra ID allows users to log in to AWS Identity Center using their Azure AD credentials. This provides a number of benefits, including improved security and convenience for users, reduced administrative overhead, and improved compliance.

SSO between AWS Identity Center and Azure Entra ID works by generating a SAML assertion from Azure AD and sending it to AWS Identity Center. AWS Identity Center then validates the SAML assertion and creates a session for the user, allowing them to access the AWS Identity Center application without having to enter their credentials again.

When provisioning users from Azure into the AWS IAM Identity Center, it's essential to seamlessly integrate them into AWS as if they were native AWS users. This involves assigning them to specific AWS accounts and granting permission sets that align with their respective roles within the organization.

## There's more...

Let us explore some more important concepts related to integrating IAM Identity Center with Microsoft Entra ID:

- **User provisioning:** This is the process of automatically creating and managing user accounts in an application or system. In the context of SSO between AWS IAM Identity Center and Microsoft Entra ID, user provisioning allows us to automatically create AWS Identity Center user accounts for users who are already in Microsoft Entra ID. This can improve security by reducing the risk of unauthorized access to our applications and resources and also reduce administrative overhead by freeing up our IT staff to focus on other tasks.
- **Group synchronization:** This is the process of keeping user groups in sync between two systems. In the context of SSO between IAM Identity Center and Microsoft Entra ID, group synchronization allows us to keep user groups in sync between Microsoft Entra ID and IAM Identity Center. This ensures that users are always assigned to the correct groups in both systems, which can improve the user experience by ensuring that they always have access to the applications and resources they need.
- **Attribute mapping:** This is the process of mapping user attributes between two systems. In the context of SSO between AWS Identity Center and Microsoft Entra ID, attribute mapping allows us to map user attributes between Microsoft Entra ID and IAM Identity Center. This is important because IAM Identity Center uses user attributes to make authorization decisions, so by mapping the correct attributes, we can ensure that users are always granted the correct access permissions.

Let us now see some additional references to help us further explore integrating IAM Identity Center with Microsoft Entra ID, especially with respect to the changes on the Azure side.

## See also

- We can explore more about automatic provisioning within AWS IAM Identity Center at <https://www.cloudericks.com/blog/understanding-automatic-provisioning-with-aws-iam-identity-center>.
- Including detailed steps and explanations for the configurations needed to be done on the Azure side would have been outside the scope of this book. We can learn more about Microsoft Entra SSO integration with AWS IAM Identity Center at <https://www.cloudericks.com/blog/understanding-microsoft-entra-sso-integration-with-iam-identity-center>.

# Additional Services and Practices for AWS Security

We have looked into many security-related concepts and services throughout this book. There are still more security services and practices that can help us make our AWS infrastructure secure. In this chapter, we will look into some of these additional services and practices that are worth paying attention to. Unlike other chapters, we will not go very deep into these services. You can refer to the links provided in the *See also* section for further learning. We will learn about some security-related managed services such as **AWS Resource Access Manager (AWS RAM)**, **AWS Systems Manager Parameter Store**, **AWS Secrets Manager**, **AWS Trusted Advisor**, and **AWS Artifact**. We will also see how we can use **Amazon Machine Images (AMIs)** and security products from AWS Marketplace.

In this chapter, we will cover the following recipes:

- Setting up and using AWS RAM
- Storing sensitive data with the Systems Manager Parameter Store
- Using AWS Secrets Manager to manage RDS credentials
- Creating an AMI instead of using EC2 user data
- Using security products from AWS Marketplace
- Using AWS Trusted Advisor for recommendations
- Using AWS Artifact for compliance reports



## Technical requirements

Before diving into the recipes of this chapter, we need to ensure we have the following requirements in place:

- We need at least an active AWS account to complete the recipes within this chapter. Unless specifically mentioned in the recipe, I will be using the `awssecbb-sandbox-1` account that we created in the *Multi-account management with AWS Organizations* recipe in *Chapter 1*, and I won't be utilizing any AWS Organizations features.
- For administrative actions, we need a user who has `AdministratorAccess` permission to the AWS account we are working with.

The code files for this book are available at <https://github.com/PacktPublishing/AWS-Security-Cookbook-Second-Edition>. The code files for this chapter are available at <https://github.com/PacktPublishing/AWS-Security-Cookbook-Second-Edition/tree/main/Chapter10>.

## Setting up and using AWS RAM

**AWS RAM** enables us to securely share AWS resources with other AWS accounts or within our AWS organization. The resources we can share include transit gateways, subnets, AWS License Manager configurations, and Amazon Route 53 resolver rules. In this recipe, we will learn to use AWS RAM to share a subnet.

### Getting ready

We need the following to successfully complete the recipe:

- A management account with AWS Organizations set up as discussed in the *Multi-account management with AWS Organizations* recipe from *Chapter 1*. I will be using the `aws-sec-cookbook-1` account that we created in that recipe.
- A member account within the organization to share resources with.
- VPC and subnets by following the *Setting up VPC plus VPC resources with minimal effort* recipe in *Chapter 5*, however, you may skip creating a NAT gateway.

## How to do it...

We can set up and use AWS RAM as follows:

1. Go to the **Resource Access Manager** service in the AWS Management Console. If we are using the service for the first time, click on **Settings** from the left sidebar, select the **Enable sharing with AWS Organizations** option, click **Save settings**, and go back to the **Resource Access Manager** dashboard.

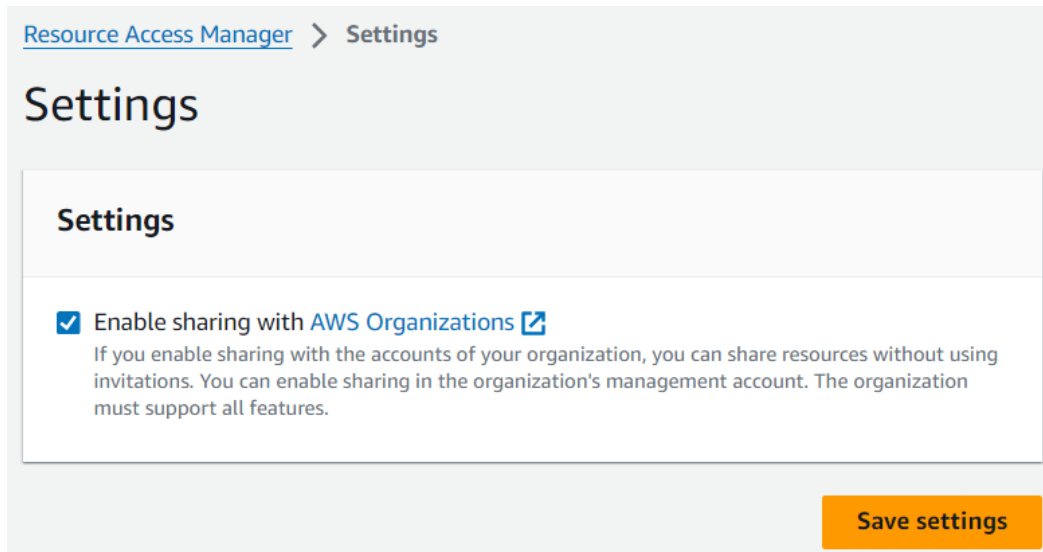


Figure 10.1 – Enable sharing with AWS Organizations

2. Click on **Create a resource share**.
3. For **Resource share name**, provide the `my-subnet-share` value.
4. In the section titled **Resources - optional**, in the dropdown for **Select resource type**, select **Subnets**, and from the subnet list, select a subnet we created for this recipe, as discussed in the *Getting ready* section.
5. Scroll down and click **Next**. We will be taken to the **Associate managed permissions** page. Leave everything as default, scroll down, and click **Next**.
6. On the **Grant access to principals** page, under the **Principals** section, select **Allow sharing only within your organization**, under **Select principal type**, choose **AWS account**, enter the account number of another AWS account, click **Add**, scroll down, and click **Next**. It will lead to the **Review and create** page. Review the details, scroll down, and click **Create resource share**.

7.
- To verify the resource share, log in to the account that we specified in *Step 6* and go to the AWS **Resource Access Manager** dashboard. Click on **Resource shares** under **Shared with me** from the left sidebar.

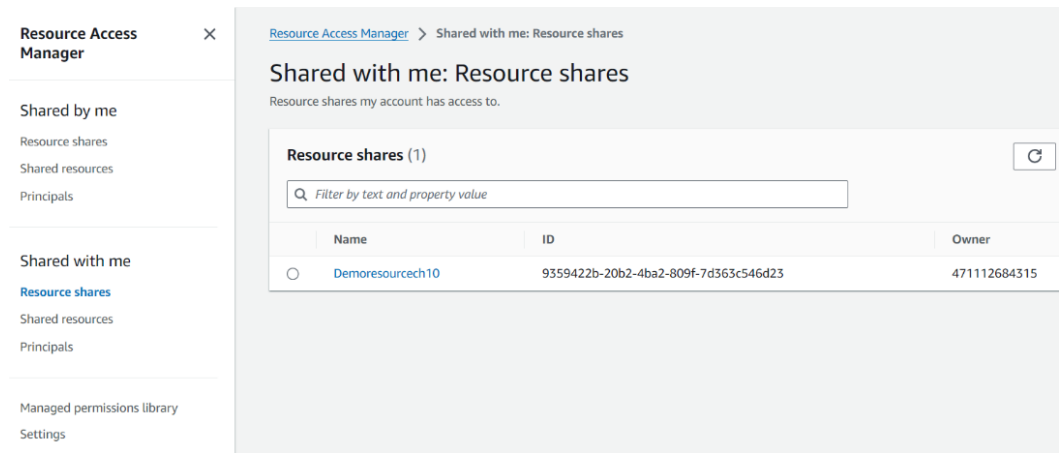


Figure 10.2 – Shared resources in our member account

We should see the subnet that was shared. We need to select the same Region where we created the shares to see the shares.

How it works...

First, we enabled sharing within our AWS organization. This can be done from the **AWS RAM** console, as we saw in this recipe. If we do not enable sharing within our AWS organization, the accounts we add will be considered external accounts, even if they are part of our organization. We added a resource share for a subnet and shared it with another account within our OU.

There’s more...

We shared a subnet resource type with AWS RAM. The following are the resource type options currently available: **Aurora DB Clusters**, **Capacity Reservations**, **CodeBuild Projects**, **CodeBuild Report Groups**, **Dedicated Hosts**, **Image Builder Components**, **Image Builder Image Recipes**, **Image Builder Images**, **License Configurations**, **Resolver Rules**, **Traffic Mirror Targets**, and **Transit Gateways**.

See also

You can read more about AWS RAM at <https://www.cloudericks.com/blog/getting-started-with-aws-ram>.

## Storing sensitive data with the Systems Manager Parameter Store

We can store data using the **Systems Manager Parameter Store** with and without encryption, and then reference it from various services without having to hardcode the data in any place. In this recipe, we will learn how to store data with encryption in an AWS Systems Manager Parameter Store and then retrieve it from an EC2 instance.

### Getting ready

We need the following to successfully complete the recipe:

- A working AWS account and a user as described in the *Technical requirements* section.
- An EC2 instance in the default VPC, within a public subnet within the VPC. For **Amazon Machine Image (AMI)**, select **Amazon Linux 2023 AMI**. For **Instance type**, select `t2.micro`. For **Key pair (login)**, select an existing one you have access to or create a new one. Under **Network settings**, make sure the value for **Auto-assign public IP** is **Enable** and **Create security group** is selected with the value for **Allow SSH traffic from** set as **Anywhere**. We learned about EC2 in *Chapter 5*.
- You would benefit from being familiar with KMS. We learned about KMS in *Chapter 3*.

### How to do it...

First, we will create a parameter in the AWS Systems Manager Parameter Store. Then, we will attach a role necessary for accessing the AWS Systems Manager from an EC2 instance. Finally, we will retrieve the parameter's information from that EC2 instance.

#### *Creating a parameter in the AWS Systems Manager Parameter Store*

We can create a Systems Manager Parameter Store parameter as follows:

1. Go to the **Systems Manager** service in the AWS management dashboard.
2. Click on **Parameter Store** from the left sidebar.
3. Click on **Create parameter**.
4. On the **Create parameter** page, for **Name**, enter `MySecureParameter`, for **Description** — **Optional**, enter `My Secure Parameter`, and select **Tier** as **Standard**.
5. For **Type**, select **SecureString**, for **KMS key source**, select **My current account**, for **KMS Key ID**, leave the default value of `alias/aws/ssm` as is, and for **Value**, enter `MySecureValue`.
6. Scroll down and click on **Create parameter**. We should get this message: **Create parameter request succeeded!**.

Next, we will create and attach a role to our EC2 instance to access the AWS Systems Manager.

## Creating and attaching a role for the AWS Systems Manager

To use the AWS Systems Manager from an EC2 instance, we need to attach a role to an EC2 instance, as follows:

1. Go to the **IAM** dashboard in the AWS Management Console.
2. Click on **Roles** from the left sidebar and click on **Create role**.
3. Under **Trusted entity type**, select **AWS service**, and select **EC2** from the list of services for **Service or use case**.
4. Scroll down and select **EC2 Role for AWS Systems Manager**. Click **Next**.

Service or use case

EC2 ▼

Choose a use case for the specified service.

Use case

- ☐ EC2  
Allows EC2 instances to call AWS services on your behalf.
- ☒ **EC2 Role for AWS Systems Manager**  
Allows EC2 instances to call AWS services like CloudWatch and Systems Manager on your behalf.
- ☐ EC2 Spot Fleet Role  
Allows EC2 Spot Fleet to request and terminate Spot Instances on your behalf.
- ☐ EC2 - Spot Fleet Auto Scaling  
Allows Auto Scaling to access and update EC2 spot fleets on your behalf.
- ☐ EC2 - Spot Fleet Tagging  
Allows EC2 to launch spot instances and attach tags to the launched instances on your behalf.
- ☐ EC2 - Spot Instances  
Allows EC2 Spot Instances to launch and manage spot instances on your behalf.
- ☐ EC2 - Spot Fleet  
Allows EC2 Spot Fleet to launch and manage spot fleet instances on your behalf.
- ☐ EC2 - Scheduled Instances  
Allows EC2 Scheduled Instances to manage instances on your behalf.

Cancel Next

Figure 10.3 – Select trusted entity

5. On the **Add permissions** pane, click on **Next**.
6. In the **Name, review, and create** page, for **Role name**, provide the `MySSMManagedInstanceRole` value, and for **Description**, provide the `Amazon SSM Managed Instance Core Role` value. Review the entire page and then click on **Create role**. We should see the message that the role was created successfully.
7. Attach this role to our EC2 instance as we saw in the *Cross-service access via IAM roles on EC2 instances* recipe from *Chapter 2*.

## Retrieving parameters from the AWS Systems Manager Parameter Store

We can retrieve the parameter value from our EC2 instance. SSH into the EC2 instance and run the following command:

```
aws ssm get-parameters --names MySecureParameter --with-decryption
--region us-east-1
```

We should get a response like the following:

```
{
  "InvalidParameters": [],
  "Parameters": [
    {
      "Name": "MySecureParameter",
      "DataType": "text",
      "LastModifiedDate": 1707898172.413,
      "Value": "MySecureValue",
      "Version": 1,
      "Type": "SecureString",
      "ARN": "arn:aws:ssm:us-east-1:636100610221:parameter/MySecurePar
parameter"
    }
  ]
}
[ec2-user@ip-172-31-8-146 ~]$ |
```

Figure 10.4 – Getting the parameter

The parameter value will be decrypted. We also have a `get-parameter` subcommand for the `aws ssm` CLI command; however, currently, the AWS-provided `AmazonEC2RoleForEC2` role does not include it. You can add the permission manually and then use `get-parameter`.

## How it works...

In this recipe, we created a parameter in the AWS Systems Manager Parameter Store and retrieved it. We can use the parameter from any service without needing to hardcode its values. Now, we can update the value for that parameter from one place.

For retrieving and decrypting the parameter value that was encrypted, we used the `get-parameters` subcommand of the `aws ssm` CLI command with the `--with-decryption` option. The default is `--no-with-decryption` if none is specified and does not decrypt the value.

We can also create parameters without encryption by setting **Type** to **String** instead of **Secure String**.

## There's more...

In this recipe, we only used one of the features of the AWS Systems Manager, namely the Parameter Store. Let's quickly go through some more important features of the AWS Systems Manager:

- AWS Systems Manager allows us to group resources such as EC2 instances, S3 buckets, and **Relational Database Service (RDS)** instances. After we've done this, we can perform actions, such as installing a patch across a group.
- We can use the parameter in the AWS Systems Manager Parameter Store from various services, such as EC2, Lambda, and CloudFormation. We can also use the parameter in a Systems Manager run command.
- The run command can be used to automate admin tasks and configuration changes across a group of EC2 machines.
- The EC2 role for a simple Systems Manager permits us to use the EC2 run command from our EC2 instances.
- We can specify EC2 target instances manually or based on a tag attached to the EC2 instances.
- When we configure a run command from the console, AWS provides us with the corresponding CLI command that we can run.
- For working with the run command, the SSM agent needs to be installed on the instances.
- We can use the run command to execute actions on on-premises systems as well.

## See also

- You can read more about the AWS Systems Manager at <https://www.cloudericks.com/blog/getting-started-with-aws-systems-manager-ssm>.
- You can read more about using KMS with the AWS Systems Manager Parameter Store at <https://www.cloudericks.com/blog/using-aws-kms-with-aws-ssm-parameter-store>.

## Using AWS Secrets Manager to manage RDS credentials

In this recipe, we will learn to use AWS Secrets Manager to manage RDS credentials. This is a more secure alternative to managing and rotating the RDS credentials manually.

## Getting ready

We need the following to successfully complete the recipe:

- A working AWS account and a user as described in the *Technical requirements* section.
- An RDS database instance created in RDS with defaults, but with the following exceptions:
  - For **Instance configuration**, select **serverless v2** to keep costs to the minimum.
  - For **Credentials Settings**, provide values for **Master username** and **Master password**, as shown in the following figure:

### ▼ Credentials Settings

#### Master username [Info](#)

Type a login ID for the master user of your DB instance.

1 to 16 alphanumeric characters. The first character must be a letter.

#### Credentials management

You can use AWS Secrets Manager or manage your master user credentials.

☐ **Managed in AWS Secrets Manager - *most secure***  
RDS generates a password for you and manages it throughout its lifecycle using AWS Secrets Manager.

☒ **Self managed**  
Create your own password or have RDS create a password that you manage.

☐ **Auto generate password**

Amazon RDS can generate a password for you, or you can specify your own password.

#### Master password [Info](#)

#### Password strength [Strong](#)

Minimum constraints: At least 8 printable ASCII characters. Can't contain any of the following symbols: / ' " @

#### Confirm master password [Info](#)

Figure 10.5 – Credentials Settings for the RDS database

### Important note

If we select the **Managed in AWS Secrets Manager** option in *Figure 10.5*, then RDS will generate a password and store it within the Secrets Manager without the need for the steps outlined in this recipe. However, we want to demonstrate storing secrets manually within this recipe and hence we selected the **Self managed** option.



## How to do it...

We can configure AWS Secrets Manager to manage the credentials of an RDS database as follows:

1. Go to the **Secrets Manager** service in the AWS Management Console.
2. From the left sidebar, click on **Secrets**. On the **Secrets** page, click on **Store a new secret**.
3. For **Secret type**, select **Credentials for Amazon RDS database**, as shown in the following figure:

The screenshot shows the 'Choose secret type' interface in the AWS Secrets Manager console. At the top, the title 'Choose secret type' is displayed. Below it, the section 'Secret type' is shown with an 'Info' link. There are five radio button options arranged in a grid:

- ☒ Credentials for Amazon RDS database
- ☐ Credentials for Amazon DocumentDB database
- ☐ Credentials for Amazon Redshift data warehouse
- ☐ Credentials for other database
- ☐ Other type of secret  
API key, OAuth token, other.

Figure 10.6 – Choosing a Secret type option in AWS Secrets Manager

4. Provide the values of **Master username** and **Master password** as **User name** and **Password** respectively.
5. For the **Encryption key** option, select the default encryption key from the dropdown. We may also create a KMS encryption key following the recipes in *Chapter 3* and use it instead, but for this recipe, we will use the default one.
6. In the **Database** section select the database we created in the *Getting ready* section and click on **Next**.
7. Provide prod/CloudericksApp/Postgres for the **Secret name** field and Access to Postgress prod database for the Cloudericks app for the **Description** field. Optionally, add any tags and click **Next**.

8. For **Configure automatic rotation**, enable **Automatic rotation**. Then, select **Schedule expression builder** for **Rotation schedule**, set **Time unit** to **Days**, and enter 30 for **Days**. For **Window duration – optional**, leave as default and select the checkbox for **Rotate immediately when the secret is stored**. The next rotation will begin on your schedule.

**Configure automatic rotation** [Info](#)  
Configure AWS Secrets Manager to rotate this secret automatically.

☒ Automatic rotation

**Rotation schedule** [Info](#)

☒ Schedule expression builder ☐ Schedule expression

Time unit 

Days

Days ▼

30

Window duration - *optional*  

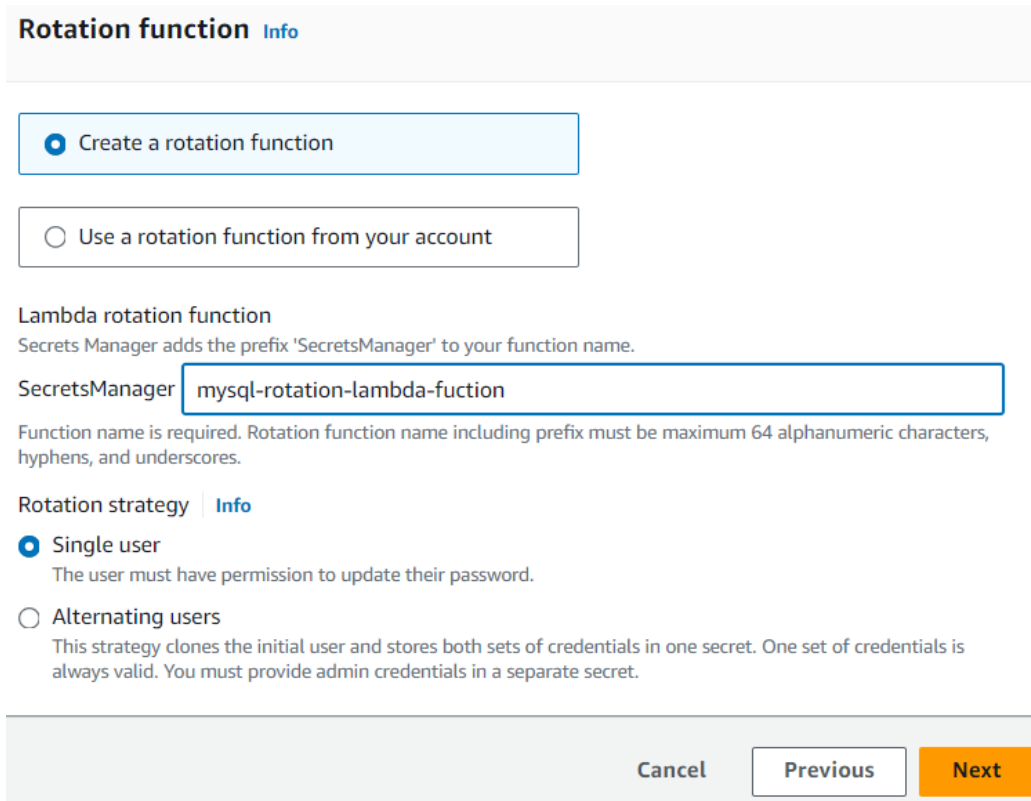
4h

Enter the time in hours.

☒ Rotate immediately when the secret is stored. The next rotation will begin on your schedule.

Figure 10.7 – Configuring automatic rotation

9. In the **Rotation function** section, select **Create a rotation function**. For **Lambda rotation function**, provide the `postgres-rotation-lambda` name. Select **Single user** for **Rotation strategy** and click **Next**.



**Rotation function** [Info](#)

☒ Create a rotation function

☐ Use a rotation function from your account

**Lambda rotation function**  
Secrets Manager adds the prefix 'SecretsManager' to your function name.

SecretsManager

Function name is required. Rotation function name including prefix must be maximum 64 alphanumeric characters, hyphens, and underscores.

**Rotation strategy** [Info](#)

☒ **Single user**  
The user must have permission to update their password.

☐ **Alternating users**  
This strategy clones the initial user and stores both sets of credentials in one secret. One set of credentials is always valid. You must provide admin credentials in a separate secret.

[Cancel](#) [Previous](#) [Next](#)

Figure 10.8 – Configuring the rotation function

10. On the **Review** page, carefully review all the details. Additionally, sample code is provided for different languages, including Java, JavaScript, C#, Python3, Ruby, Go, and Rust, demonstrating how to retrieve the secret from our application, as shown in *Figure 10.9*. Finally, click **Store**.

## Sample code

Use these code samples to retrieve the secret in your application.

[Java](#)[JavaScript](#)[C#](#)[Python3](#)[Ruby](#)[Go](#)[Rust](#)

```
1 // Use this code snippet in your app.
2 // If you need more information about configurations or implementing
  the sample
3 // code, visit the AWS docs:
4 // https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/home
  .html
5
6 // Make sure to import the following packages in your code
7 // import software.amazon.awssdk.regions.Region;
8 // import software.amazon.awssdk.services.secretsmanager
  .SecretsManagerClient;
9 // import software.amazon.awssdk.services.secretsmanager.model
  .GetSecretValueRequest;
10 // import software.amazon.awssdk.services.secretsmanager.model
  .GetSecretValueResponse;
```

Java

Line 32, Column 63

✖ Errors: 0

⚠ Warnings: 0

[Download AWS SDK for Java](#)

Cancel

Previous

Store

Figure 10.9 – Sample code to retrieve secrets from Secrets Manager

11. Once we get the success message, go to the new secret to verify the details.
12. Scroll down and go to the **Overview** tab, and then click **Retrieve secret value** to view our secret. If you want to edit them, you can do so using the **Edit** button.

If we want to delete a secret, we can go to the secret, click on the **Actions** dropdown, and click on **Delete secret**. We must specify a waiting period of between 7 and 30 days before the secret will be deleted.

## How it works...

In this recipe, we stored credentials for our RDS database in Secrets Manager. For the secret type, we selected **Credentials for Amazon RDS database**. The following are the other secret type options currently available in the console: **Credentials for Redshift cluster**, **Credentials for DocumentDB database**, **Credentials for other database**, and **Other type of secrets** (e.g., API key).

We enabled automatic key rotation with a duration of 30 days. We can also select 60 days or 90 days, or provide a custom period up to 365 days. I selected the default encryption key for this recipe. Instead, you can use a KMS key you created. We learned about KMS keys in *Chapter 3*.

After configuring a secret, our application can make an API call to Secrets Manager to retrieve the secret programmatically. While storing the secret, AWS provides us with sample code for different languages to retrieve the secret. Currently, there are sample codes for Java, JavaScript, C#, Python3, Ruby, Go, and Rust.

We saw a warning that the first rotation would happen immediately upon storing this secret. Therefore, if any of our applications are still using hardcoded credentials and are not updated to use the APIs to get the latest credentials, those applications will fail.

## There's more...

AWS Secrets Manager may look like AWS Systems Manager Parameter Store. Let's quickly compare Secrets Manager with Parameter Store:

- Secrets Manager is primarily used for storing database credentials, API keys, and SSH keys. Parameter Store is primarily used for storing license codes, configuration data, user-defined parameters, and database strings, and is less commonly used for passwords too.
- AWS Secrets Manager is charged per secret per month and per API call. AWS Systems Manager Parameter Store does not charge for standard parameters but charges us for advanced parameters based on the number of advanced parameters stored and per API interaction.
- Secrets Manager has built-in integration with RDS databases. Secrets Manager supports the built-in rotation of secrets for RDS. It also supports the rotation of non-RDS databases using custom Lambda functions.
- Parameter Store is integrated with AWS Systems Manager.

## See also

Read more about the AWS Secrets Manager and Parameter Store comparison at <https://www.cloudericks.com/blog/comparing-aws-secrets-manager-and-parameter-store>.

## Creating an AMI instead of using EC2 user data

In this recipe, we will create an AMI with a web server and then launch an instance from that AMI. Instances from AMIs have faster boot times than instances with the same configuration defined through EC2 user data. In the *Launching an EC2 instance with a web server using user data* recipe from *Chapter 5*, we used EC2 user data to update our operating system and set up a simple web server at launch.

### Getting ready

We need the following to successfully complete the recipe:

- A working AWS account and a user as described in the *Technical requirements* section.
- A launched EC2 instance, launched following the *Launching an EC2 instance with a web server using user data* recipe in *Chapter 5*.

### How to do it...

We can create an AMI from an EC2 instance as follows:

1. Go to the **EC2** service in the console.
2. Click on **Instances**, select our instance, click **Actions**, expand **Image and templates**, and then click **Create image**.
3. On the **Create image** screen, provide **Image name** and **Image description - optional**. Use the defaults for the other parameters and click on **Create image**.
4. If we go to the AMIs list, our AMI should display its initial status as **Pending**. Once the status changes to **Available**, create a new instance from this AMI. While launching the instance, choose our AMI from the **My AMI** tab.

### How it works...

In this recipe, we created an AMI from an EC2 instance. Information related to the launch of an instance, including any organization-specific configuration, can be saved into an AMI. We used Amazon Linux 2023 as our base AMI within this recipe and hence used commands that are specific to Amazon Linux 2023. If you are using Amazon Linux 2, you can also use the same commands.

We did a similar configuration in *Chapter 5* using EC2 user data. Instances from AMIs have faster boot times than instances with the same configuration defined through EC2 user data. This is because we can have packages preinstalled in an AMI, whereas we need to install them at launch with user data.

## There's more...

Let's quickly go through some important concepts related to AMIs:

- Multiple EC2 instances can be launched from a single AMI.
- AMIs are specific to a Region, but you can copy them across Regions. While launching an instance, we can choose between the AMIs suggested by Amazon, our own AMIs, AWS Marketplace AMIs, and community AMIs. We can also filter the list based on additional parameters.
- We should only use public AMIs that we trust. We can check ratings for AMIs before using them.
- AMIs are stored in S3. Hence, we will be charged based on S3 pricing, which is also dependent on our free tier eligibility and usage. However, we won't be able to see the AMI or its bucket from the S3 console.
- By default, AMIs are private for our account and Region. We can make our AMIs public for other AWS accounts to use or sell them through AWS Marketplace.

## See also

- You can read more about AMIs at <https://www.cloudericks.com/blog/getting-started-with-amis-for-ec2-instances-in-aws>.
- Read more about AMI hardening at <https://www.cloudericks.com/blog/ami-hardening-for-ensuring-security-and-stability-in-our-aws-environment>.

## Using security products from AWS Marketplace

In this recipe, we will learn to use various **security products** from AWS Marketplace. Many third-party companies will install and configure their products and solutions on EC2 instances and provide them as AMIs in AWS Marketplace. Marketplace AMIs can be considered EC2 instances with pre-configured software. Alternatively, we can also buy the products directly from these vendors and do the configurations on our own.

## Getting ready...

We need a working AWS account, and a user as described in the *Technical requirements* section.

## How to do it...

We can find and use security-related AMIs from AWS Marketplace as follows:

1. Go to the **EC2** service on the dashboard.
2. Click on **Instances** from the left sidebar and click **Launch instances**.

3. Under **Application and OS Images (Amazon Machine Image)**, click on **Browse more AMIs**.

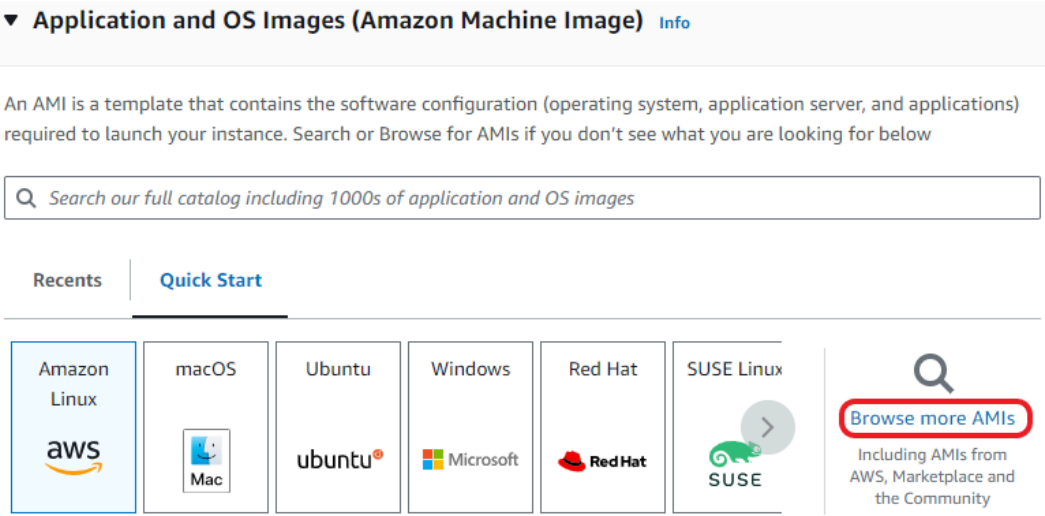


Figure 10.10 – Browsing AMIs from AWS Marketplace

4. Go to the **AWS Marketplace AMIs** tab and search for **security** in the search bar. As of this writing, 10719 AMIs are available.

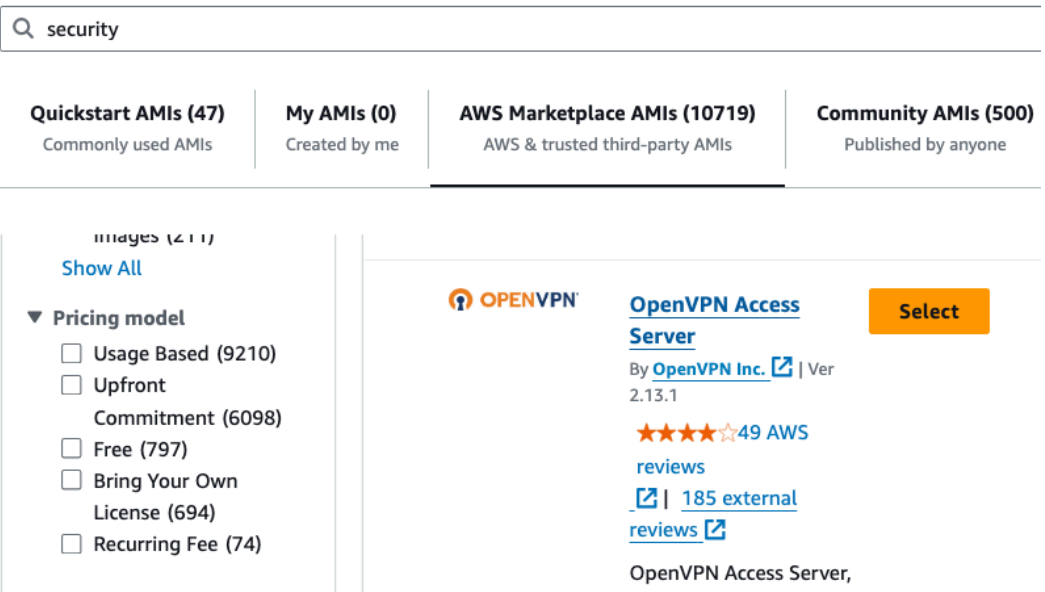


Figure 10.11 – AWS Marketplace AMIs



We can further filter results based on parameters such as **Operating System**, **Software Free Trial**, **Software Pricing Plans**, and **Support** from the left sidebar. Once we decide on a product, we can follow the recipes within *Chapter 5*, to complete launching the instance.

## How it works...

AWS may not be able to provide all the security products we need. Many third-party companies have developed products and solutions that complement AWS services for security and compliance. We saw how to find such AMIs in AWS Marketplace. Once we decide on a product, we can launch an instance with that AMI. For more details about the particular product we selected, we can refer to the respective product's documentation.

## There's more...

**Network packet inspection**, also referred to as **deep packet inspection (DPI)**, inspects packet headers and data contents of packets to detect non-compliant data, viruses, spam, and so on, and can take actions such as blocking and logging. It combines the functionalities of a traditional firewall with an **intrusion detection system (IDS)** or an **intrusion prevention system (IPS)**.

**AWS Web Application Firewall (WAF)**, the firewall service in AWS, can check for known exploits such as SQL injection, cross-site scripting, and so on. However, AWS cannot do a complete network packet inspection and lacks the functionality of an IDS and IPS. We can, however, use solutions from AWS Marketplace. There are solutions provided by vendors including Alert Logic, Trend Micro, McAfee, Palo Alto Networks, and Cisco Systems, among others.

## See also

You can read about security products in AWS Marketplace at <https://www.cloudericks.com/blog/getting-started-with-aws-marketplace>.

## Using AWS Trusted Advisor for recommendations

In this recipe, we will learn to use Trusted Advisor. **Trusted Advisor** is an online tool in AWS that provides recommendations related to cost optimizations, performance, security, fault tolerance, and service limits.

## Getting ready

We need a working AWS account, and a user as described in the *Technical requirements* section.

## How to do it...

We can use Trusted Advisor as follows:

1. Go to the **Trusted Advisor** service in the AWS Management Console. We should see the **Recommendations** categories and basic recommendations on the dashboard landing page, as shown in the following figure:

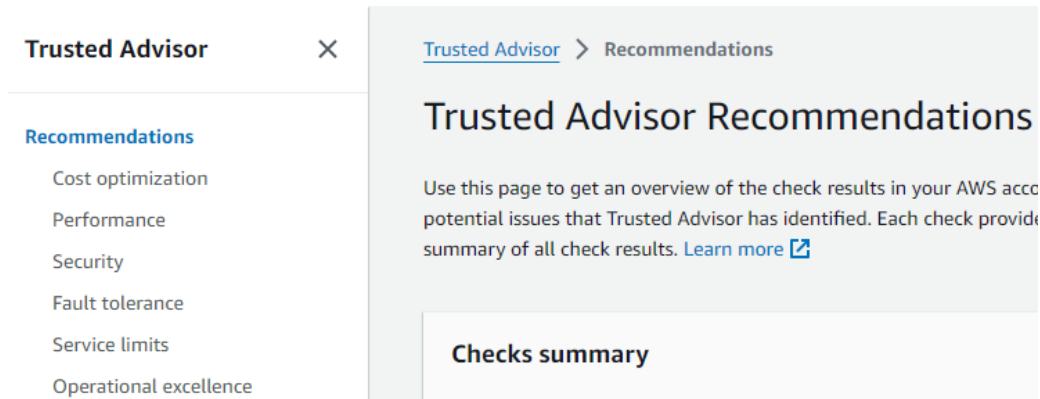


Figure 10.12 – The Trusted Advisor dashboard

2. Click on **Security** from the left sidebar to see recommendations related to security.

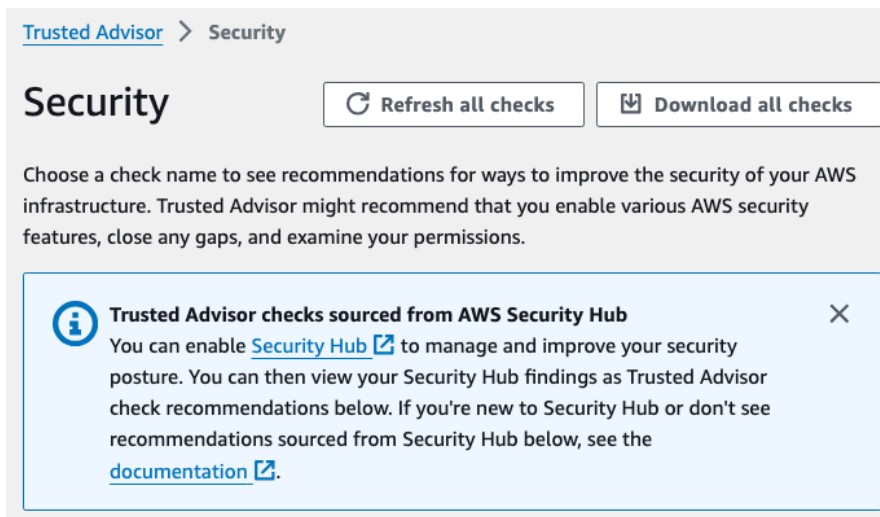


Figure 10.13 – The Security recommendation page

3. Click on **Download all checks** to download all checks or click on the download button for a recommendation to download the details of that recommendation.

4. Click on **Refresh all checks** to download all checks or click on the refresh button for a recommendation to refresh the details of that recommendation.
5. Click on **Service limits** from the left sidebar to see recommendations related to service limits such as services that use more than 80% of service quota.

**Important note**

As of this writing, except for **Security** and **Service limits**, all other recommendations are only available after upgrading. We can upgrade after clicking on **Recommendations** from the left sidebar or after going to any of the recommendation category options in the left sidebar except **Security** and **Service limits**.

## How it works...

Trusted Advisor provides recommendations related to cost optimizations, performance, security, fault tolerance, service limits, and operational excellence. There are two service levels: the Basic plan and the fully Trusted Advisor. The Basic plan is free and covers core checks and recommendations. Fully Trusted Advisor functionality is available for the Developer, Business, and Enterprise AWS support plans.

## There's more...

The Trusted Advisor Basic plan currently has no recommendations available under cost optimization, performance, fault tolerance, and operational excellence. For these categories, recommendations are available only with the fully Trusted Advisor. For the **Security** and **Service limits** categories, some recommendations come with both the Basic plan as well as the full Trusted Advisor.

## See also

You can read more about Trusted Advisor at <https://www.cloudericks.com/blog/understanding-aws-trusted-advisor>.

## Using AWS Artifact for compliance reports

In this recipe, we will learn to use AWS Artifact. AWS Artifact is a free, self-service portal for accessing AWS's compliance reports. AWS Artifact can be used to access AWS's security and compliance reports and select online agreements.

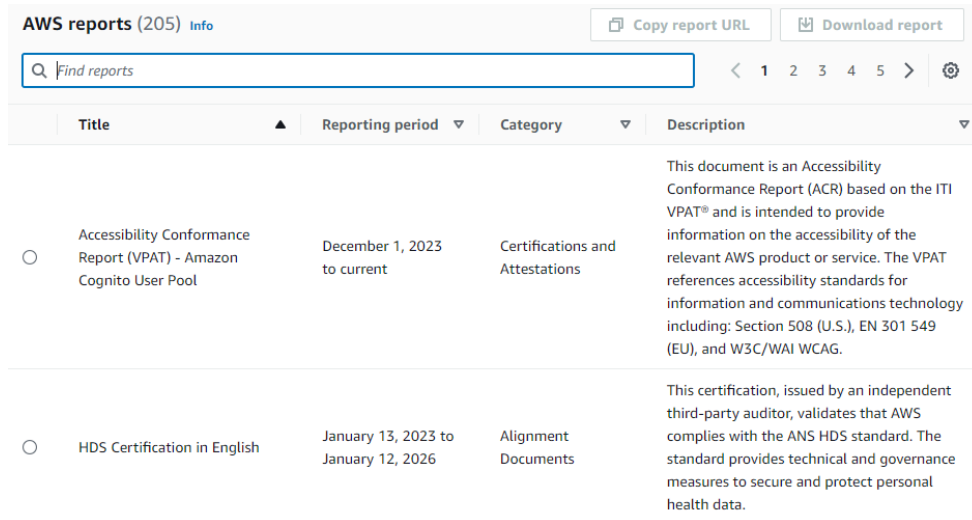
## Getting ready

We need a working AWS account, and a user as described in the *Technical requirements* section.

## How to do it...

We can use AWS Artifact as follows:

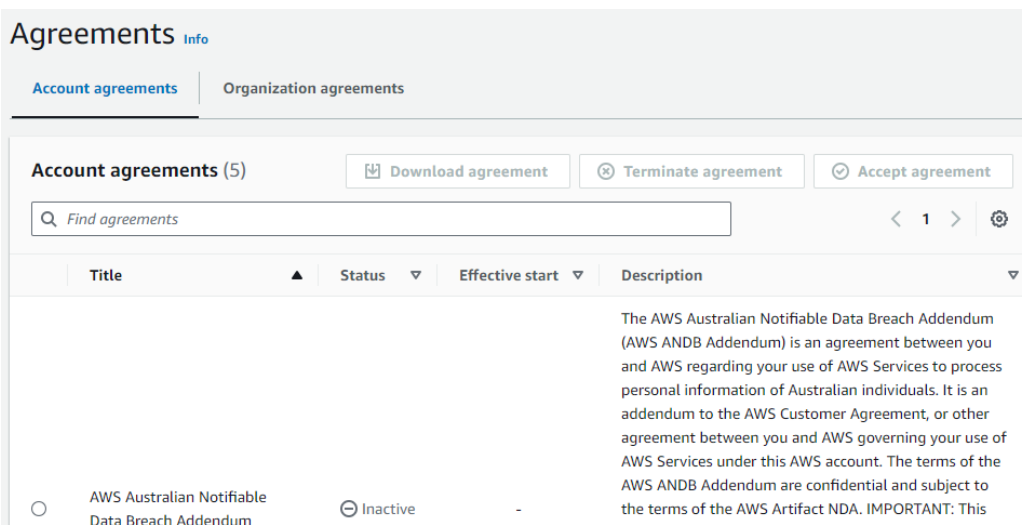
1. Go to the **AWS Artifact** service in the console.
2. Click on **Reports** in the left sidebar to see the available reports.



	Title ▲	Reporting period ▼	Category ▼	Description ▼
<input type="radio"/>	Accessibility Conformance Report (VPAT) - Amazon Cognito User Pool	December 1, 2023 to current	Certifications and Attestations	This document is an Accessibility Conformance Report (ACR) based on the ITI VPAT® and is intended to provide information on the accessibility of the relevant AWS product or service. The VPAT references accessibility standards for information and communications technology including: Section 508 (U.S.), EN 301 549 (EU), and W3C/WAI WCAG.
<input type="radio"/>	HDS Certification in English	January 13, 2023 to January 12, 2026	Alignment Documents	This certification, issued by an independent third-party auditor, validates that AWS complies with the ANS HDS standard. The standard provides technical and governance measures to secure and protect personal health data.

Figure 10.14 – Reports from AWS Artifact

3. Click **Agreements** in the left sidebar to see **Account agreements** and **Organization agreements**.



Agreements Info				
Account agreements   Organization agreements				
Account agreements (5)				
<input type="button" value="Download agreement"/> <input type="button" value="Terminate agreement"/> <input type="button" value="Accept agreement"/>				
<input type="text" value="Find agreements"/>				
	Title ▲	Status ▼	Effective start ▼	Description ▼
<input type="radio"/>	AWS Australian Notifiable Data Breach Addendum	<input type="radio"/> Inactive	-	The AWS Australian Notifiable Data Breach Addendum (AWS ANDB Addendum) is an agreement between you and AWS regarding your use of AWS Services to process personal information of Australian individuals. It is an addendum to the AWS Customer Agreement, or other agreement between you and AWS governing your use of AWS Services under this AWS account. The terms of the AWS ANDB Addendum are confidential and subject to the terms of the AWS Artifact NDA. IMPORTANT: This

Figure 10.15 – Agreements

**Tip**

We can click on the **Organization agreements** tab from an organization's master account to manage agreements for our master account and all member accounts in our organization.

4. Select any agreement and click on the **Download agreement** button to download the agreement. If an **Accept NDA to download report** message appears, we need to select the **I have read and agree to the all the terms of the NDA** checkbox.

Accept NDA to download report

This confidential document is subject to the terms of the AWS Artifact Nondisclosure Agreement (AWS Artifact NDA). You must agree to the terms by checking the box at the end of this document before you can download the selected artifact. The AWS Artifact NDA is not intended to replace any other NDA between you and Amazon. If you have a separate NDA with Amazon that applies to the information provided in AWS Artifact, then that separate NDA will apply instead of the AWS Artifact NDA (see Section 11 of the Artifact NDA).

AWS Artifact Nondisclosure Agreement

Click **Accept NDA and download**.

☒ I have read and agree to the all the terms of the NDA.

Print NDA

Cancel

Accept NDA and download

Figure 10.16 – Accepting and downloading the report

**How it works...**

We've learned to use AWS Artifact for checking compliance reports. The following are some of the reports currently available with AWS Artifact: Accessibility Conformance Report (VPAT) – Amazon Cognito User Pool, HDS Certification in English, ABS Cloud Computing Implementation Guide 2.0 – Workbook, Accessibility Conformance Report (VPAT) – Amazon API Gateway, Accessibility Conformance Report (VPAT) – Amazon AppFlow, Accessibility Conformance Report (VPAT) – Amazon AppStream 2.0, Accessibility Conformance Report (VPAT) – Amazon Simple Queue Service, Accessibility Conformance Report (VPAT) – Amazon Simple Notification Service (SNS), and Accessibility Conformance Report (VPAT) – Amazon Simple Email Service (SES).

## There's more...

We saw many AWS services within this book to help us secure our infrastructure on AWS. However, that is not an exhaustive list. AWS also constantly adds more services and features. Let's quickly go through some more services related to security:

- **Amazon Detective** is a service that can help in finding the root cause of potential security issues by analyzing and visualizing security data. As of this writing, this service is in preview.
- **AWS Control Tower** helps us to set up and govern a multi-account AWS environment based on best practices related to security and compliance. End users can provision new accounts following the company-wide compliance policies centrally established, and the cloud administrators can see a complete overview of the landing zone. A landing zone is a container for all of our OUs, accounts, users, and other resources that need to be compliant. Landing zones should be in a non-member account of an organization.
- **AWS License Manager** helps us manage licenses from third-party vendors, such as Microsoft, Oracle, and SAP when we bring them to AWS. We can control their usage and even set customized rules on their usage for different groups of users.
- **AWS Personal Health Dashboard** is a service available to customers with the Premium support plan to monitor, manage, and optimize their AWS environment.
- **AWS Well-Architected Tool** is a management service based on the **AWS Well-Architected Framework**. We can define workloads against current AWS best practices and this tool will provide guidance on how to improve our cloud architectures.

## See also

You can read about AWS Artifact at <https://www.cloudericks.com/blog/getting-started-with-aws-artifact>.



# Index

## A

### **access**

providing, to AWS accounts 31-33

### **Access Control Lists (ACL) 135**

versus IAM policies and bucket policies 152, 153

### **access, restricting to Amazon Simple Storage Service origin**

reference link 244

### **account**

creating, from CLI 19-21

customer-managed keys,  
sharing 120, 121, 124

### **account alias**

setting up 3, 6, 7, 11

### **accounting 12, 255**

### **AD Connector 354**

### **alerting 255**

### **Amazon API Gateway 220**

### **Amazon Athena**

reference link 281

using, to query CloudTrail  
logs in S3 277-280

### **Amazon Cloud Directory 354**

### **Amazon CloudFront 220**

### **Amazon CloudWatch 255**

### **Amazon CloudWatch Events 297**

### **Amazon Cognito 324**

custom authentication challenges 350

multi-factor authentication (MFA) 350

reference link 338

social IdPs 350

user pool 339, 354

user pools, working with 324-338

### **Amazon Cognito Identity SDK 345**

### **Amazon Cognito User Pools 354**

### **Amazon Detective 387**

### **Amazon Elastic Container Registry (ECR) 310**

### **Amazon EventBridge**

setting up, to monitor findings

from Access Analyzer 320

setting up, to monitor findings

from GuardDuty 297

working with 264-268

### **Amazon GuardDuty 291**

concepts 296

configuring, to aggregate findings from  
multiple accounts 298, 299

IPs, blacklisting 293-295

IPs, whitelisting 293-295

reference link 298



- setting up 292, 293
- using 292, 293
- Amazon Inspector 291**
  - reference link 311
  - setting up 306-310
  - using 306-310
- Amazon Machine Images (AMIs) 152**
  - concepts 380
  - creating, from EC2 instance 379
  - reference link 380
- Amazon Macie 291**
  - concepts 305
  - reference link 306
  - setting up 300-304
  - using 300-304
- Amazon Resource Name (ARN) 48**
- Amazon S3 Bucket 339**
- Amazon Simple Notification Service (SNS) 325**
  - reference link 338
- Amazon Verified Permissions 338**
  - reference link 338
- AMI hardening**
  - reference link 380
- API keys 300**
- app client**
  - with ALLOW\_USER\_PASSWORD\_AUTH flow 339
- Appliance User (AU) 132**
- application load balancers (ALBs)**
  - concepts 235
  - using, with TLS termination at ELB 228-234
- asymmetric encryption 93**
- Attribute-Based Access Control (ABAC) 37**
- attributes for access control 37**
- auditing 255**
- authentication 12**
  - access credentials 12
- Authentication, Authorization, and Accounting (AAA) 1**
- authorization 12**
- auto-remediation 255**
- availability 12**
- Availability Zones (AZs) 178**
- AWS**
  - interacting, ways 24
- AWS access keys**
  - use cases 13
- AWS access portal URL 33**
- AWS accounts**
  - access, providing to 31-33
  - creating 17, 18
  - IAM, setting up 4-6
  - moving, under Sandbox OU 17, 18
- AWS Artifact**
  - reference link 387
  - using, for compliance reports 384-386
- AWS Budgets**
  - reference link 14
- AWS Certificate Manager (ACM) 217, 331**
  - used, for creating SSL/TLS certificate 220-224
- AWS CLI**
  - customer-managed IAM policy via 64-67
  - destination account, setting up via 82, 83
  - reference link 41
  - source account, setting up via 83, 84
- AWS CLI documentation, for organizations**
  - reference link 25
- AWS CLI subcommands, for AWS Organizations 24**
- AWS CLI v2 136**
- AWS Cloud**
  - reference link 14
- AWS CloudFormation 220**

**AWS CloudHSM vs AWS KMS**

reference link 133

**AWS CloudTrail 255****AWS CloudTrail log 22****AWS Config 255**

reference link 289

setting up 284-289

using 284-289

**AWS Console**

presigned URL, generating from 154, 155

**AWS Control Tower 387**

reference link 25

**AWS Elastic Beanstalk 220****AWS Firewall Manager**

reference link 254

**AWS Global Accelerator 232****AWS GuardDuty**

findings, exporting 299, 300

**AWS IAM Identity Center**

references 364

**AWS IAM policy evaluation logic 55, 56****AWS Key Management Service****Keys (SSE-KMS)**

features 170

**AWS KMS**

keys, importing into 105, 106

reference link 101

**AWS KMS condition keys**

reference link 120

**AWS KMS key policies**

reference link 120

**AWS KMS, key policies vs grants**

reference link 120

**AWS KMS key rotation 108**

reference link 108

**AWS KMS service 100, 101****AWS KMS with External Key Material**

reference link 106

**AWS KMS within AWS Cloud**

reference link 125

**AWS Lambda 310****AWS License Manager 387****AWS-managed keys 94****AWS Managed Microsoft AD 354****AWS managed policy**

permission set, creating with 30, 31

**AWS Management Console 136**

customer-managed IAM policy,

creating 45-49, 62-64

destination account, setting up via 77, 78

IAM policies, attaching to IAM

group via 49-51

key material, uploading 104, 105

roles, switching via 79-82

source account, setting up via 78, 79

**AWS Marketplace**

reference link 382

security products, using from 380-382

**AWS Organizations**

creating, from management console 15, 16

multi-account management

with 14, 15, 21, 22

reference link 25

**AWS Organizations service 23****AWS-owned keys 94****AWS Personal Health Dashboard 387****AWS policies and permissions**

reference link 56

**AWS Policy Generator 56****AWS policy types**

ACLs 55

identity-based policies 54

permissions boundaries 54

resource-based policies 54

SCPs 55

session policies 54

**AWS policy variables**

reference link 60

**AWS Resource Access Manager  
(AWS RAM) 366**

reference link 368

resource type 368

setting up 366-368

using 366-368

**AWS resources**

accessing, with identity pools 338-350

**AWS Secrets Manager**

using, to manage RDS credentials 372-378

**AWS Secrets Manager versus  
Parameter Store 378**

reference link 378

**AWS Security Hub**

concepts and services 317

reference link 318

setting up 311-317

using 311-317

**AWS services, integrated with  
AWS Organizations**

reference link 25

**AWS shared responsibility model 12**

reference link 14

**AWS Shield**

reference link 254

**AWS Shield Advanced 254****AWS Shield Standard 254****AWS Simple AD**

Amazon EC2 354

Amazon RDS 354

Amazon WorkSpaces 354

cost-effective directory services 354

integration, with AWS services 353

Lightweight Directory Access

Protocol (LDAP) 353

references 354

SSO 353

used, for creating lightweight

directory solution 350-353

user and group management 353

**AWS SSO 1, 75****AWS Systems Manager**

features 372

reference link 372

**AWS Systems Manager Parameter Store**

parameter, creating 369

parameters, retrieving from 371

role, attaching 370

role, creating 370

used, for storing sensitive data 369, 371

**AWS Systems Manager Session Manager 194****AWS Trusted Advisor**

reference link 384

using, for recommendations 382-384

**AWS VPCs**

concepts 181

subnets 181, 182

**AWS Well-Architected Tool 387**

## B

**bare VPC**

auto-assign public IPv4

address, enabling 184

creating 183, 184

IGW, configuring 185

IGW, creating 185

route table, configuring 185

route table, creating 185

subnet, creating 184

working 186

**billing alarm**

setting up 3, 7-11

**blacklisted IPs (Threat IPs) 294**

**Bring Your Own Key (BYOK)** 101

**Bucket policy** 56, 135

## C

**CanonicalUser** 143

**Cedar** 338

**Center for Internet Security (CIS)** 310

**Certificate Authority (CA)** 217

**certificates** 215

**certificate signing request**  
(CSR) 128, 217-219

**cipher text** 93

**Classless Inter-Domain Routing**  
(CIDR) 48, 178

**client apps, Cognito**

confidential client 336

custom app 336

public client 336

**client-side encryption** 163

**Cloud Development Kit (CDK)** 24

**CloudFormation template**

usage 281-284

**CloudFront distribution**

adding, to an S3 bucket with custom  
domain and ACM certificate 244-246

adding, to S3 bucket with default CloudFront  
domain and certificate 238-244

**CloudHSM**

reference link 133

user types 132

**CloudHSM cluster**

activating 125, 129-132

creating 125-132

initializing 125-132

**CloudShell**

reference link 25

**CloudTrail**

CloudWatch, integrating with 281-284

reference link 271, 277

trail, creating 271-277

**CloudTrail API event logs** 271

**CloudTrail log events**

reading and filtering 268-271

**CloudTrail logs, in S3**

Amazon Athena, using to query 277-280

**CloudWatch alarms**

working with 258-262

**CloudWatch custom metrics**

reference link 263

**CloudWatch Events** 264

reference link 268

**CloudWatch log group** 208

creating 208, 263

reference link 264

**CloudWatch metrics**

working with 258-262

**Command Line Interface (CLI)**

account, creating from 19-21

OU, creating from 19-21

presigned URL, generating from 155, 156

roles, switching via 84, 85

**compliance reports**

AWS Artifact, using for 384-386

**compliance validation for AWS IAM**

reference link 14

**conditional keys**

key policies, using with 114-119

**confidentiality** 12

**Confidentiality, Integrity, and**  
**Availability (CIA)** 1, 255

**confused deputy** 86

**Content Delivery Networks (CDNs)** 215

**continuous monitoring** 255

**cross-account CloudTrail logging** 280

reference link 281

**cross-account replication** 164

**Cross-Region Replication (CRR)** 164

**cross-service**

accessing, via IAM roles on

EC2 instances 87-90

**Crypto Officer (CO)** 132

**Crypto User (CU)** 132

**customer-managed IAM policy**

creating 45, 51

creating, from AWS Management

Console 45-49, 61, 62

via AWS CLI 64-67

via AWS Management Console 62-64

**customer-managed keys** 94

sharing, across accounts 120-124

**customer-managed policies**

creating, in IAM Identity Center 60, 67, 68

**customer master keys (CMKs)** 95

**Customer-Provided Keys (SSE-C)** 163

**custom Macie data identifier**

adding 303, 304

**custom permission set** 60

## D

**data encryption, S3** 164

concepts 169

SSE, with SSE-C 168, 169

SSE, with SSE-KMS 166, 167

SSE, with SSE-S3 165

**data integrity** 12

**data keys** 100

**Data Security Standard (DSS)** 296

**decryption** 93

**deep packet inspection (DPI)** 382

**default VPC settings** 181

**delegated administrator** 61

**Denial of Service (DOS) attack** 296

**destination account**

setting up, via AWS CLI 82, 83

setting up, via AWS Management

Console 77, 78

**Distributed Denial of Service**

(DDoS) 215, 254

**Domain Name System (DNS)**

reference link 224, 247

**Domain Validated (DV) certificates** 220

**Dynamic Host Configuration**

Protocol (DHCP) traffic 210

**DynamoDB** 207

## E

**EBS instance**

reference link 195

**EC2 instance**

Amazon Machine Images (AMIs),

creating from 379

connecting, over SSH 191-194

launching, with web server using

user data 187-191

reference link 195

**EC2 instance profiles and IAM roles**

reference link 92

**Elastic Block Storage (EBS)** 295

**Elastic Compute Cloud (EC2)** 257

**Elastic Load Balancing (ELB)** 220

**Elastic Network Interface (ENI)** 207

**ELB target group**

creating 224-228

**encryption** 93

creating 224-228

reference link 101

**encryption algorithm** 93

**encryption keys** 93  
**ephemeral port** 198  
**essential tools to Secure IAM**  
    reference link 321  
**Extended Validation (EV) certificate** 219  
**external ID** 86  
    reference link 87  
**external key**  
    key configuration, creating for 102-104  
**external key material (BYOK)**  
    used, for creating keys 101-105

## F

**Federal Risk and Authorization  
    Management Program (FedRAMP)** 12  
**Federated IdPs, with Amazon Cognito**  
    reference link 350  
**findings, exporting**  
    reference link 300  
**firewalls** 215

## G

**Glacier Vault Lock** 162  
    setting up 163  
**Google Authenticator** 3  
**governance**  
    centralizing, in AWS Organizations  
        with SCPs 71-74  
**grants in AWS KMS**  
    reference link 114  
**groups** 11

## H

**hardware MFA device** 3  
**hardware security module (HSM)** 94, 125

**HTTPS**  
    enabling, for web server on  
        EC2 instance 217-219  
**HTTPS listener, creating for  
    Application Load Balancer**  
    reference link 235

## I

**IAM Access Analyzer**  
    functionalities 320  
    using, to inspect unused access 318, 319  
**IAM cross-account role switching**  
    75, 76, 85, 86  
**IAM Identity Center** 136  
    customer-managed policies, creating 60-68  
    enabling 26, 27  
    group, creating 26-30  
    reference link 41  
    Single Sign-On (SSO) with 25, 36-39  
    used, for configuring SSO for  
        AWS CLI 34, 35  
    user, creating 26-30  
    user management with 25, 37-39  
**IAM Identity Center, in AWS CLI**  
    used, for logging in and out for SSO 36  
**IAM Identity Center integration,  
    with Microsoft Entra ID**  
    attribute mapping 363  
    group synchronization 363  
    user provisioning 363  
**IAM permission boundaries**  
    setting, for IAM entities 68-70  
**IAM policies** 43, 52, 56  
    attaching, to IAM group via AWS  
        Management Console 49-51  
    elements 52  
    example 52  
    structure 52, 53

**IAM policy variables** 56

using 57-60

**IAM roles** 43, 90

concepts 91, 92

**IAM roles on EC2 instances**

cross-service access via 87-90

**IAM users and groups** 13**identity account** 85**identity account architecture** 75, 76, 85, 86

reference link 87

**Identity and Access Management (IAM)**

setting up 3, 11

setting up, for AWS accounts 4-6

versus ACLs and bucket policies 153

**identity pools**

using, to access AWS resources 338-350

**Identity Provider (IdP)** 37, 337**Inspector Classic** 311

reference link 311

**International Organization for  
Standardization (ISO)** 12**Internet Gateway (IGW)** 174**intrusion detection system (IDS)** 382**intrusion prevention system (IPS)** 382**K****key**creating and giving, permission  
to other account 121, 122creating, with external key material  
(BYOK) 101, 102, 105using, as administrator user  
from account 2 122using, as non-admin user from  
account 2 123, 124**key administrator** 95**key configuration**

creating, for an external key 102-104

**Key Management Service (KMS)** 94, 163

keys, creating 95-99

**key material** 99

generating, with OpenSSL 104

uploading, in AWS Management  
Console 104, 105**key policies**

using 119, 120

using, with conditional keys 114-119

**key rotation**

in KMS 106, 107

**key user** 95**KMS grants**used, for granting permissions  
programmatically 109-112**KMS, using with AWS Systems  
Manager Parameter Store**

reference link 372

**L****Lambda** 256**lightweight directory solution**

creating, with AWS Simple AD 351-353

**load balancers** 215**load balancing in AWS**

reference link 228

**log records**

reference link 210

**M****Macie alerts**notifications, configuring with Amazon  
EventBridge and SNS 305, 306**main route table** 179**management account** 21, 61

**management console**

AWS Organizations, creating from 15, 16  
 Organizational Units (OUs),  
 creating from 16, 17

**master account 21****member account**

creating, as delegated administrator 40

**metric filters 281****Microsoft Entra ID, as IdP within AWS**

AWS IAM Identity Center,  
 setting up 358-360

AWS IAM Identity Center users,  
 verifying 361, 362

Azure configuration, continuing 360

Azure, configuring for IAM Identity  
 Center integration 357

metadata file, downloading  
 from AWS 355-357

using 354, 355

working 363

**multi-account management**

with AWS Organizations 14, 15, 21, 22

**multi-factor authentication (MFA) 3, 60, 285****multiple accounts, in GuardDuty**

findings, aggregating from 298, 299

**multi-region keys 100****N****Network Access Control Lists (NACLs) 247**

reference link 205

working with 199-205

**Network Address Translation**

(NAT) gateway 174, 180

concepts 213

configuring 210-212

creating 211

reference link 213

settings 211

**network load balancer (NLB)**

using, with TLS termination at EC2 235-237

**network packet inspection 382****non-repudiation 12****O****on-demand scans 310****One Time Password (OTP) 335****Open ID Connect (OIDC) protocols 337****OpenSSL**

URL 101

used, for generating key material 104

**OpenSSL Alongside, using Default**

LibreSSL on MacOS

reference link 102

**Open Threat Exchange (OTX) CSV 295****Organizational Units (OUs) 14, 71**

creating, from CLI 19-21

creating, from management console 16, 17

**Organization Validated (OV) certificate 219****Origin Access Control (OAC) 244****P****Parameter Store**

versus AWS Secrets Manager 378

**Payment Card Industry Data Security**

Standard (PCI DSS) 12

**Payment Card Industry (PCI) 296****permissions**

granting 113, 114

granting, programmatically with  
 KMS grants 109-112

revoking 113, 114

**permissions boundaries for IAM entities**

reference link 71

**permissions boundary 23**



- permission set** 97, 362
  - creating, with AWS managed policy 30, 31
- Personally Identifiable Information (PII)** 300
- plain text** 93
- policies** 11
- policies and permissions within AWS**
  - reference link 120
- policy editor** 79
- policy store** 338
- Precrypto Officer (PRECO)** 132
- primary PCO** 132
- private subnets** 174, 178
- Protected Health Information (PHI)** 300
- public subnets** 174, 178, 182
- publish/subscribe messaging service** 256

## R

- RDS credentials**
  - managing, with AWS Secrets Manager 372-378
- Relational Database Service (RDS)** 372
- Role-Based Access Control (RBAC)** 37
- roles** 11
  - switching, via AWS Management Console 79-82
  - switching, via CLI 84, 85
- root account** 3
- route tables** 174, 179

## S

- S3 access logs** 164
- S3 Access Points** 163
- S3 ACLs** 56
  - concepts 151, 152
  - working with 146-152

- S3 bucket** 271
- S3 bucket policy**
  - creating 136-139
  - ListBucket Access, granting for IAM user from CLI 142-145
  - ListBucket and GetObject access, granting from console 139-142
- S3 encryption** 163
- S3-Managed Keys (SSE-S3)**
  - features 170
- S3 object locking**
  - used, for protecting files 157-162
- S3 presigned URLs**
  - creating 153
  - presigned URL, generating from AWS Console 154-156
- S3 replication** 164
- S3 versioning**
  - used, for protecting files 157-163
- SAML 2.0-enabled applications** 37
- Secure Shell (SSH)**
  - using 187
  - using, for EC2 connection 191-193
- Secure Sockets Layer (SSL)**
  - reference link 220
- security assessments** 306
- security groups**
  - configuring 197, 198
  - creating 195
- security key** 3
- security posture, monitoring with Security Hub standards and controls**
  - reference link 318
- security products**
  - using, from AWS Marketplace 380-382
- Security Token Service (STS)** 86
- Serverless Application Model (SAM)** 24
- Server-Side Encryption (SSE)** 136

**Service Control Policies (SCPs) 23, 71**

reference link 74

used, for centralizing governance in  
AWS Organizations 71-74

**Service Organization Control (SOC) 12****Services integrated with AWS****Certificate Manager**

reference link 224

**SHA\_256 105****Simple Email Service (SES) 257****Simple Mail Transfer Protocol (SMTP) 257****Simple Notification Service**

(SNS) 10, 56, 255

**Simple Queue Service (SQS) 56, 256****Simple Storage Service (S3) 271**

data encryption 164

securing, with CloudFront 238

securing, with TLS 238

**Single Sign-On (SSO) 25, 351**

configuring, for AWS CLI with

IAM Identity Center 34, 35

logging in and out, with IAM Identity  
Center in AWS CLI 36

with IAM Identity Center 25, 37-39

**SMS 256****SNS topic**

creating, to send emails 256, 257

**Software Bill of Materials (SBOM) 310****Software Development Kits (SDKs) 24****source account**

setting up, via AWS CLI 83, 84

setting up, via AWS Management  
Console 78, 79

**SSE-C**

features 170, 171

**SSL/TLS certificate**

creating, with ACM 220-224

**sso-admin namespace and commands**

reference link 68

**statement identifier (Sid) 119****Stream Control Transmission**

Protocol (SCTP) 204

**Structured Query Language (SQL) 275****Structured Threat Information**

Expression (STIX) 295

**symmetric encryption 93****symmetric key 95**

## T

**TLS termination, at EC2**

network load balancer, using with 235-237

**TLS termination, at ELB**

application load balancers (ALBs),

using with 228-234

**TLS termination for load balancers in AWS**

reference link 237

**traffic, routing to CloudFront**

distribution using Route 53

reference link 247

**Transmission Control Protocol**

(TCP) ports 194

**Transport Layer Security (TLS) 215, 257**

reference link 220

**trust policy 82**

## U

**U2F-compliant device 3****unused access**

IAM Access Analyzer, used

to inspect 318, 319

**User Datagram Protocol (UDP) 204****user management**

with IAM Identity Center 25, 37-39

**user pools**

working with 324-338

**users 11****V****Virtual Private Cloud (VPC) 52, 173**

interface endpoints 207

**Virtual Private Network (VPN) 173****VPC Endpoints (S3 Gateway) 174, 180****VPC flow logs**

concepts 210

configuring 208

reference link 210

using 209

**VPC gateway endpoint 207**

reference link 208

used, for connecting to S3 205-207

**VPC plus VPC resources**

setting up, with minimal effort 174-180

**W****Web Application Firewall (WAF) 382**

reference link 254

using 247-253

**web server, on EC2 instance**

HTTPS, enabling for 217-219

**Well-Architected Framework 387****whitelisted IPs (Trusted IPs) 294****Write Once, Read Many (WORM) 157****Y****YubiKey Universal 2nd Factor (U2F) 3**



packtpub.com

Subscribe to our online digital library for full access to over 7,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

## Why subscribe?

- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Improve your learning with Skill Plans built especially for you
- Get a free eBook or video every month
- Fully searchable for easy access to vital information
- Copy and paste, print, and bookmark content

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at [packtpub.com](http://packtpub.com) and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at [customercare@packtpub.com](mailto:customercare@packtpub.com) for more details.

At [www.packtpub.com](http://www.packtpub.com), you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

# Other Books You May Enjoy

If you enjoyed this book, you may be interested in these other books by Packt:



## **Building and Automating Penetration Testing Labs in the Cloud**

Joshua Arvin Lat

ISBN: 978-1-83763-239-8

- Build vulnerable-by-design labs that mimic modern cloud environments
- Find out how to manage the risks associated with cloud lab environments
- Use infrastructure as code to automate lab infrastructure deployments
- Validate vulnerabilities present in penetration testing labs
- Find out how to manage the costs of running labs on AWS, Azure, and GCP
- Set up IAM privilege escalation labs for advanced penetration testing
- Use generative AI tools to generate infrastructure as code templates
- Import the Kali Linux Generic Cloud Image to the cloud with ease



## Securing Cloud PCs and Azure Virtual Desktop

Dominiek Verham, Johan Vanneuville

ISBN: 978-1-83546-025-2

- Become familiar with Windows 365 and Microsoft Azure Virtual Desktop as a solution
- Uncover the security implications when company data is stored on an endpoint
- Understand the security implications of multiple users on an endpoint
- Get up to speed with network security and identity controls
- Find out how to prevent data leakage on the endpoint
- Understand various patching strategies and implementations
- Discover when and how to use Windows 365 through use cases
- Explore when and how to use Azure Virtual Desktop through use cases

## Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit [authors.packtpub.com](https://authors.packtpub.com) and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

## Share Your Thoughts

Now you've finished *AWS Security Cookbook*, we'd love to hear your thoughts! If you purchased the book from Amazon, please [click here](#) to go straight to the Amazon review page for this book and share your feedback or leave a review on the site that you purchased it from.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

---

## Download a free PDF copy of this book

Thanks for purchasing this book!

Do you like to read on the go but are unable to carry your print books everywhere?

Is your eBook purchase not compatible with the device of your choice?

Don't worry, now with every Packt book you get a DRM-free PDF version of that book at no cost.

Read anywhere, any place, on any device. Search, copy, and paste code from your favorite technical books directly into your application.

The perks don't stop there, you can get exclusive access to discounts, newsletters, and great free content in your inbox daily

Follow these simple steps to get the benefits:

1. Scan the QR code or visit the link below



<https://packt.link/free-ebook/9781835081891>

2. Submit your proof of purchase
3. That's it! We'll send your free PDF and other benefits to your email directly



